EUROGRAPHICS 2014 / B. Lévy and J. Kautz
(Guest Editors)

Volume 33 (2014), Number 2

Pose Partitioning for Multi-resolution Segmentation of Arbitrary Mesh Animations

Andreas A. Vasilakis † and Ioannis Fudos ‡

Dept. of Computer Science & Engineering, University of Ioannina, Greece

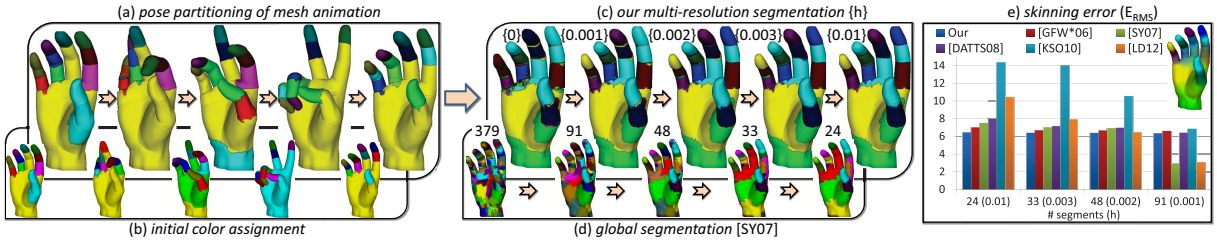

Figure 1: (a) A consistent colorization of five representative (b) random painted partitionings computed from each pose. (c) Our segmentations are superior in terms of (e) skinning error when compared to prior art ((d) illustrated segmentations of [SY07]) in most of the testing resolutions. Skinning weights computed from the smallest segmentation are also shown.

Abstract

We present a complete approach to efficiently deriving a varying level-of-detail segmentation of arbitrary animated objects. An over-segmentation is built by combining sets of initial segments computed for each input pose, followed by a fast progressive simplification which aims at preserving rigid segments. The final segmentation result can be efficiently adjusted for cases where pose editing is performed or new poses are added at arbitrary positions in the mesh animation sequence. A smooth view of pose-to-pose segmentation transitions is offered by merging the partitioning of the current pose with that of the next pose. A perceptually friendly visualization scheme is also introduced for propagating segment colors between consecutive poses. We report on the efficiency and quality of our framework as compared to previous methods under a variety of skeletal and highly deformable mesh animations.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Segmentation of mesh animations, despite being a new research field when compared to static mesh partitioning [Sha08], has become a key issue in a number of computer graphics applications. Animation compression [AS07], skinning mesh animations [JT05, KSO10, LD12], skeleton extraction [SY07, DATTS08, HTRS10], deformation transfer [LWC06] and ray tracing [GFW*06] are representative applications enabled by partitioning a *deforming mesh sequence*: a surface mesh with dynamic geometry but fixed connectivity. Mesh animations can be roughly divided into two categories: (i) *off-line* which consists of a fixed number of stored consecutive animated meshes and (ii) *real-time* which is either streamed from a shared distributed virtual environment or dynamically generated from interactive manipulation of a deformable object.

While the output depends on the type of application, the main goal of segmentation is to partition the animated mesh into regions with similar motion characteristics. Many geometric properties have been proposed for defining the feature

† abasilak@cs.uoi.gr
‡ fudos@cs.uoi.gr

space. Significant features, which form a dense region in feature space, can be detected by one of the numerous available clustering techniques. From now on, we denote such approaches as *global segmentation* methods, because they work with average motion measures that represent the degree of deformation during the entire animation sequence. However, these methods are rather limited in cases where the feature vector space is either (i) *flat*: unable to separate regions with similar feature values (zero-variance) or (ii) *anisotropic*: one or more features dominate the others due to higher variation.

Regardless the clustering criteria, current methods focus on detecting segments with mostly rigid behavior, failing in partitioning correctly highly-deformable objects. In addition, the segmentation output highly dependents on a large set of *parameters* that should be determined a priori. Finally, the entire process cannot be carried out when the user requests a different segmentation resolution or the mesh sequence is *modified* [CH12]: either being (i) subjected to pose editing operations or (ii) augmented with additional poses that did not exist in the original mesh animation.

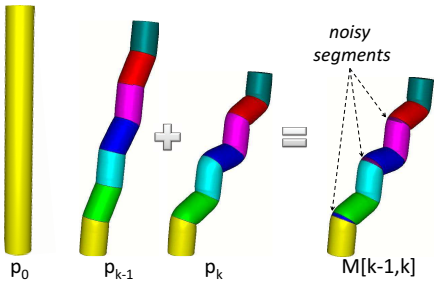

Figure 2: Illustrating the extra segments created when two consecutive pose partitionings are successfully merged.

We introduce a generic framework for efficiently generating multi-resolution segmentations that account for both articulated and highly-deformable mesh animations. Based on the observation that only a limited part of the surface region is modified from frame-to-frame, we build an *over-segmentation* by combining precomputed per-pose partitionings. The desired segmentation resolution is dynamically determined by the user by applying a fast refinement process which aims at cleaning insignificant or "noisy" segments created when successive partitionings are merged (see Figure 2). Contrary to global segmentation methods, our pipeline can handle both off-line and real-time mesh animations by exploiting different merging strategies (see Figure 3(a),(b)). Despite the independent per pose partitioning, a consistent segmentation is maintained over time (also known as *variable segmentation* [ACH*13]) by merging the partitioning of the current pose with that of the following one (see Figure 3(c)). Thus, each segmentation is similar to the one in the previous step and accurately reflects the new data arriving [CKT06]. Finally, a novel visualization scheme is introduced that provides perceptual consistency between consecutive poses.

2. Related Work

A large variety of 3D mesh segmentation algorithms using different partition criteria has been introduced the last few

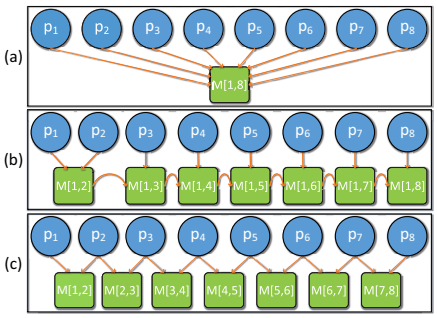

Figure 3: Three different ways of joining a sequence of partitionings: (a) *off-line*, (b) *real-time* and (c) *variable* segmentations. M$[x,y]$ denotes the over-segmentation between $[p_x,\ldots,p_y]$ poses.

years in the literature [Sha08]. Most of them are targeted on partitioning static objects into either meaningful volumetric components (*part-type* segmentation) [DGGV08] or surface patches (*surface-type* segmentation) [JKS05], depending on the application. Recently, several approaches adapted previous segmentation algorithms to work with 3D deforming meshes exploiting the analysis of the motion information.

[DIHF12] utilized **multi-source region growing** algorithm based on similarity of statistical *variability* characteristics to favor grouping between surface regions. Similarly, [LWC06] partitioned a mesh sequence into clusters with similar rigid motion by growing feature clusters based on *geodesic* and *deformation* distances. [KMD*07, KSO10] used *uniform distribution* and *deformation gradients* [SP04] to initialize their skinning decomposition, respectively.

[LD12] replaced previous clustering method with **K-means** and used bone *transformation matrices* as assignment attribute. [AS07] further employed K-means based on the local similarity between the *trajectories* in a cluster-defined coordinate system to assist their compression method. When the number of resulting clusters is unknown a priori, **mean-shift** clustering was applied based on *rotation matrices* [JT05] and *geometric invariant* feature vectors [LXL*12] to segment animated objects into near-rigid components.

[SY07] and [WPP07] derived a **bottom-up hierarchical** clustering by merging (initially per facet assigned) clusters until one node is remained based on *rigid* and *affine transformation* metrics, respectively. Conversely, [GFW*06] applied a **top-down hierarchical** approach to break mesh down into sub-meshes with similar *affine motion*. Starting from one cluster which represents the entire object, a partition is created by segmenting it into two or more components. On the other hand, [ACH*13] proposed to incrementally refine the final segmentation as a new pose arrives by splitting current components into parts which present consistent *rigid* motion.

Recent work by [DATTS08] and [HTRS10, ACH*13] exploited **spectral clustering** to segment a deformable mesh into approximately rigidly moving groups using *euclidean distance* and *rotation angle* similarity metrics, respectively. Moreover, [FKY*10] used spectral clustering for effective curvilinear feature and deformation discontinuity detection.

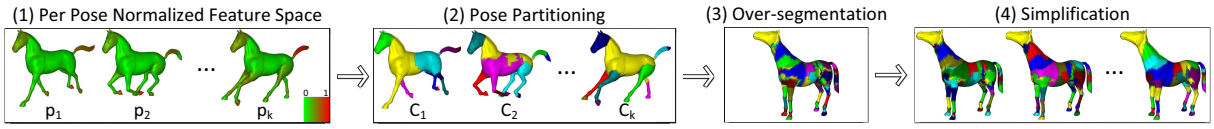

Figure 4: Diagram of the proposed pipeline.

Finally, [WB10] produced a near-rigid segmentation finding the **minimum spanning tree** of the mesh's dual graph weighted by *dihedral angles* of neighbor faces.

Similarly to our method, *co-segmentation* techniques [HFL12] may be adjusted to consistently partition a sequence of animated poses. However, they perform relative slow, they do not support pose editing as well as they are limited to work only on quasi-rigid animations with fixed frames.

3. Framework Overview

We present a general method to efficiently segmenting arbitrary mesh animations involving two main steps. An over-segmentation is first constructed by combining a set of individual partitionings corresponding to each input pose of the animation sequence. Partitioning of each pose is precomputed in parallel using any of the numerous available feature measures and clustering methods (Section 3.1). A *pose partitioning feature* (usually a deformation measure) is computed from an animation pose and a reference pose. A progressive simplification process is subsequently applied to refine the segmentation graph guided by a temporal-coherent area-aware edge collapsing technique (Section 3.2). An example ilustrating the overall information flow and the individual steps of our method is shown in Figure 4.

3.1. Pose partitioning-aware over-segmentation

Let M be a mesh of fixed connectivity represented by a graph $M = (V,E)$, where V is the set of vertices ($|V| = n$) and E is the set of edges. Let $p_i : E \mapsto R^3$ be an assignment of 3D values to the vertices that corresponds to pose i in an animation sequence $A = (p_0, p_1, p_2, \ldots, p_k)$ with k animation poses $p_1, p_2, \ldots, p_k$ and a rest-pose p_0. Let $C_i(M), i = 1, \ldots, k$ be a partitioning (clustering) of V, such that each subset corresponds to a connected induced sub-graph of M. $C_i(M) = \{C_i^0, \ldots, C_i^l\}$ represents the resulted clusters based on pose p_i.

Definition 3.1 We define a clustering animation vector (**CAV**) for each vertex $v \in V$ such that $cav(v) = (m_1, m_2, \ldots, m_k)$, where $cav(v)[j] = m_j \in 1, \ldots, |C_j(M)|$ is the cluster index that v belongs to in pose p_j. Similarly, $cav(v)[i, j]$ is the vector of cluster indices where v belongs to in poses $p_i, p_{i+1}, \ldots, p_j$.

Definition 3.2 Let NS_i (neighbor similarity) for animation A be a binary relation between vertices. We say that for two vertices $u\, NS_i\, v$ if and only if $cav(v)[1,i] = cav(u)[1,i] \wedge ((u,v) \in E \vee u = v)$. Clearly, NS_i is reflexive and symmetric for all $i \in [1, \ldots, k]$. By taking the transitive closure of NS_i, denoted by TNS_i we have an equivalence relation. The equivalence relation partitions V in equivalence classes.

Definition 3.3 The *over-segmentation* $OS(A)$ of A over $C_i(M), i = 1, \ldots, k$ is a partitioning of V in equivalence classes called segments based on the transitive closure of the binary relation NS_k. Then, we denote by $segment(v,i)$, the segment (equivalence class) where v belongs based on the equivalence relation TNS_i. For $i = k$, we obtain segments of the final over-segmentation $OS(A)$. For other $i < k$, we obtain the corresponding segment that is produced by the over-segmentation of $OS(p_0, p_1, \ldots, p_i)$.

Property 3.4 $segment(v, j) \subseteq segment(v, i),\ i \leq j$

Proof For $i = j$, the two segments are identical, since we have equivalence classes. For $i < j$, let vertex $u \in segment(v, j)$, then there is a path between u and v, and for every vertex w in the path it holds $cav(w)[1, j] = cav(v)[1, j]$. Then, for all vertices w in this path from u and v, it holds $cav(w)[1,i] = cav(v)[1,i]$. So, $u \in segment(v,i)$. □

Immediate from the property above is that $segment(u_1, i) = segment(u_2, i)$, for all vertices $u_1, u_2 \in segment(v, j), i < j$. Thus, we may denote this new segment by $segment(segment(v, j), i)$.

To algorithmically obtain the segments of the over-segmentation (equivalence classes of $OS(A)$), we can easily prove that this is equivalent to detecting for a vertex v the maximal connected induced sub-graph of M where v belongs and all its vertices have the same CAV. To compute the over-segmentation, we consider each vertex and perform a pruned breadth-first-search to detect connected vertices with the same CAV. During this process, we mark each edge so that we will not visit it again. This takes time $O(|E|)$ which for regular non-manifold objects is $O(|V|) = O(n)$. The details of this algorithm are shown in Algorithm 1. Figure 5 illustrates the CAV generation for all clusters of the resulted over-segmentation created from three pose partitionings, where $cav(S_j)$ corresponds to the CAV of segment S_j (vertices belong to same segment have the same CAV).

3.2. Progressive decimation of over-segmentation

Following the generation of the over-segmentation, we perform a cleaning with parameter $h \in [0,1]$ called *p2p-cleaning* (pose-to-pose cleaning) starting from pose p_k, then for pose p_{k-1} towards the first animation pose p_1. Each cleaning operation $R_i(h)$ on pose p_i is based on the following **reduction rule**: Given a pose p_i and a pair of segments (S_A, S_B), S_B absorbs S_A if and only if the following holds for S_A:

$$a(S_A) \leq h \cdot a(C_i^{cav(S_A)[i]}) \quad (1)$$

$$a(S_A) \leq h \cdot a(segment(S_A, i-1)) \quad (2)$$

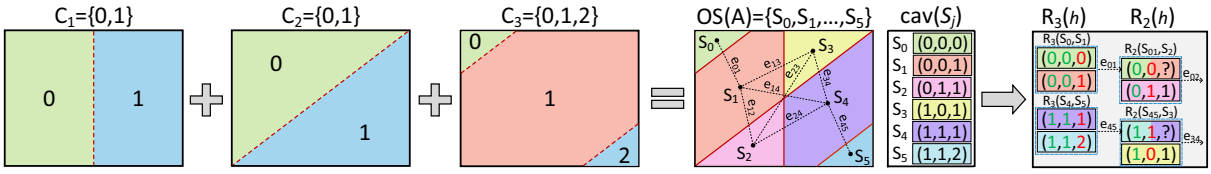

Figure 5: Illustrating the CAV for each vertex and segment created from joining three pose clusterings C_1, C_2 and C_3. The over-segmentation graph OS(A) is decimated only at the edges $\{e_{01}, e_{45}, e_{02}, e_{34}\}$ which satisfy the temporal-coherent reduction rule.

Algorithm 1 Over-segmentation(A)

```
1:  OS(A) ← ∅; VL ← list of all vertices in V;
2:  k_max ← 0; ∀v ∈ V : v.segment ← −1;
3:  Mark all edges as not visited;
4:  while VL ≠ ∅ do
5:      v ← VL.next;
6:      if v.segment == −1 then
7:          S_{k_max} ← {v}; v.segment ← k_max;
8:          k ← k_max; k_max ← k_max + 1;
9:      else
10:         k ← v.segment;
11:     end if
12:     for each not visited edge e_i ← (v, v_i) do
13:         Mark edge e_i as visited;
14:         if cav(v) == cav(v_i) then
15:             S_k ← S_k ∪ {v_i};
16:             v_i.segment ← k;
17:         end if
18:     end for
19:     Remove vertex v from list VL;
20: end while
21: OS(A) = {S_0, S_1, ..., S_{k_max−1}};
```

and S_B is a segment such that,

$$S_B \in N(S_A) \wedge cav(S_A)[1, i-1] = cav(S_B)[1, i-1] \quad (3)$$
$$\wedge\, cav(S_A)[i] \neq cav(S_B)[i]$$
$$a(S_B) > a(S_A) \quad (4)$$

where $N(S_j)$ are all neighbor segments of S in the final over-segmentation and $a(S_j)$ is the area of segment S_j. Note that after the reduction, the area of the new cluster is considered for the purposes of reductions in this pose to be this of the absorbing cluster S_B.

The conditions for S_B (3 and 4) state that S_B is the largest neighbor of S_A (larger than S_A) in the over-segmentation and they belong to the same cluster in all poses from p_1 to p_{i-1} and to a different cluster in pose p_i. This means that S_A and S_B were split to represent separate segments in pose p_i. Thus, we need to check whether one of them was erroneously created due to small cluster border differences, and should therefore be absorbed by the other. This is checked by the two conditions for S_A. The first condition ensures that S_A is small as compared to the cluster that contains it in pose p_i, thus it is not a significant part of a segment at pose p_i. The second condition states that S_A is small as compared to the superset of segment S_B in pose p_{i-1} and they have been split into two or more parts in pose p_i. This corresponds to a segment of the over-segmentation of the animation $(p_0, p_1, \ldots, p_{i-1})$ and at this phase is a candidate group of clusters (including S_A and S_B) to become one (or more) independent meaningful segment(s) after cleaning.

This reduction rule exploits temporal coherency, which means that we can perform an educated reverse pose-to-pose cleaning of non meaningful clusters preserving useful deformation information. We have explored global rules (not per pose) and we have observed that they tend to favor larger clusters without respecting other cluster characteristics. This fact makes them inappropriate for mesh segmentation of animation sequences. Figure 5 illustrates the possible reduction steps applied to an over-segmentation graph when our p2p-cleaning is employed. Note that from all potential collapsing edges, only four satisfy our history-based condition.

The details of this algorithm is shown in Algorithm 2. The initialization takes k steps. At each step it takes $O(n)$, to initialize the areas, to build the graph of adjacent segments, and to reconstruct the partial over-segmentation. Then, with careful updating of visited edges in the segment neighbor graph we are able to carry out each step at time $O(r)$, where $r = |OS(A)|$. Thus, the cleaning process takes time $O(k(n+r))$. The following substantiates the correctness of the p2p-cleaning process.

Lemma 3.5 The p2p-cleaning process will have a unique result in a finite number of steps.

Proof Since we reduce the number of segments by one at each step, the p2p-cleaning process may take at most $|OS(A)|$ reduction steps overall for all poses. Next, we shall prove that at each pose, we obtain a unique segmentation which is not affected by the order in which we apply the reductions. We can think of the cleaning process as a rewrite system, which will have a unique eventual result. More specifically, we will prove that the reduction rules are compliant to the *Church Rosser* property. Thus, if at some step of the cleaning process at pose p_i, we have a partially cleaned over-segmentation L and we have two candidates: a reduction $R_i(S_A, S_B)$ yielding a new over-segmentation Q and a reduction $R'_i(S'_A, S'_B)$ yielding a new over-segmentation Q', then it suffices to prove that there is a sequence of reductions from Q and a sequence of reductions from Q' so that we obtain the same partially cleaned over-segmentation configuration U (see Figure 6(top,left)). Figure 6 illustrates how a portion of the over-segmentation at pose p_i is build when the C_i (painter with red) breaks the existing segments of the over-segmentation of the animation $(p_0, \ldots, p_{i-1})$ (painted with green) into one or more components. Since this is always a partitioning, we distinguish among the following cases:

1. S_A, S_B, S'_A and S'_B are four disjoint sets (for example $S_A =$

c_c, $S_B = c_d$, $S'_A = c_b$ and $S'_B = c_e$), in which case we can always carry out the other reduction, and reach the same U.
2. If we have two pairs of identical sets then this is the same reduction, since this is only feasible when $S_A = S'_A$ and $S_B = S'_B$.
3. There is just one pair of identical sets. Then, we cannot have $S_A = S'_A$ since only one of the two reduction would have been eligible. If we have $S_B = S'_B$ (for example $S_A = c_b$, $S_B = S'_B = c_e$, $S'_A = c_e$), then if S'_A is absorbed by $S'_B = S_B$ then the new cluster will also absorb S_A and vice versa. It remain to consider the case were $S_A = S'_B$ (or $S'_A = S_B$). For example $S_B = c_e$, $S_A = S'_B = c_d$ and $S'_B = c_c$. If we carry out first $R_i(S_A, S_B)$, the new area of the merged cluster will be the one of S_B which by definition is larger than S_A, thus the other reduction $R'_i(S'_A, S'_B)$ will still be eligible. If $R'_i(S'_A, S'_B)$ is carried out first then the new merged cluster will have the area of $S_B = S'_A$, therefore it will be still eligible to be absorbed by S'_B.

□

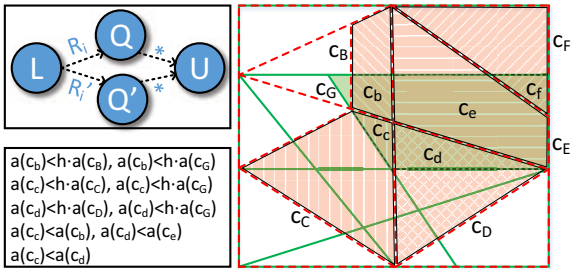

Figure 6: Illustrating the correctness proof: In this example, we close up at the reduction that will restructure the green cluster c_G, which belongs to the over-segmentation until pose p_{i-1}, when is decomposed by the red partitioning of p_i. For small h, segments c_e and c_d will absorb c_f and c_c.

Algorithm 2 p2p-Cleaning($OS(A), h$)

```
1: for p ← k, 1 do
2:     OS(A).computeArea();
3:     for each S_A ∈ OS(A) do
4:         if Clean(S_A, C_p^{cav(S_A)[p]}, h, p) then
5:             maxArea ← a(S_A);
6:             for S_B ∈ N(S_A) do
7:                 if maxArea < a(S_B) and cav(S_A)[1, p−1] ==
   cav(S_B)[1, p−1] and cav(S_A)[p] ≠ cav(S_B)[p] then
8:                     maxArea ← a(S_B);
9:                 end if
10:            end for
11:            if maxArea > a(S_A) then
12:                S_B.Copy(S_A);          ▷ neighbors,vertices,faces
13:                comment: Leave a(S_B) unchanged
14:                OS(A).Remove(S_A);
15:            end if
16:        end if
17:    end for
18: end for

19: function CLEAN(S_A, S_P, h, p)
20:    if a(S_A) > h · a(S_P) then
21:        return false;
22:    end if
23:    S_L = new List<Vertex>();
24:    comment: The following is realized with a breadth
25:    first search from S_B (similar to Algorithm 1)
26:    for each v ∈ segment(S_A, p−1) do
27:        S_L.Add(v);
28:    end for
29:    return a(S_A) ≤ h · a(S_L);
30: end function
```

4. Applications

4.1. Smooth Visualization of Cluster Transitions

Since clusters may vanish, shift or arise when moving through time, we introduce a perceptually friendly visualization scheme to propagate as much as possible the segment colors between consecutive frames. A user should perceive the transition from one frame to the next one avoiding if possible to encounter totally different coloring of clusters. We follow a strategy that aims at covering a high distribution of the color space minimizing the possibility of "close" colors be assigned to neighbor clusters. The algorithm starts by initially painting the partitioning of the first pose, followed by a propagation of the cluster colors from pose to pose.

Rest-pose coloring. A breadth-first traversal is applied by picking a random cluster as root node. At each node visit, we set the next color from the palette shown in Figure 7 that does not conflict with an assigned color from its neighborhood. A 2-ring neighborhood can also be used to increase color distribution. The palette is an *RGB color wheel* with 12 divisions: an illustrative organization of color hues around a circle that consists of the primary:{red,green,blue}, secondary:{cyan,yellow,magenta} and tertiary colors:{colors between primary and secondary ones}. Note that complementary colors lie opposite to each other in the color sphere. In case of color overflow (chromatic number of the cluster graph > 12), a larger ring of colors should be used.

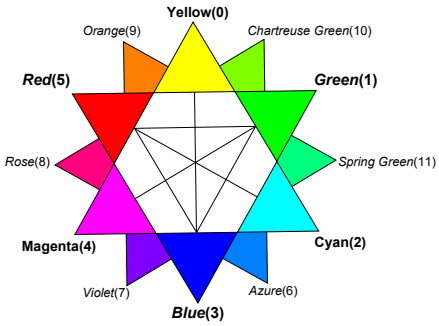

Figure 7: RGB color wheel. Color order is shown in brackets.

Pose-to-pose color propagation. A new *future-cluster* initially computes how much area covers from each of its overlapping *past-clusters* in linear time. A breadth-first traversal is afterwards applied picking as root the cluster with the largest covering area. Each future-cluster inherits the color of the first of its past-clusters (sorted by the overlapping area)

that is not covered by any of the rest of the future-clusters. Note that this color must not conflict with any previously assigned color from its neighborhood. If a future-cluster covers only one past-cluster and cannot inherit any color, we assign it the next available color from the palette. Otherwise, we use a mixed color of its two largest covering past-clusters.

To avoid the following problems that arise when mixing neighbor cluster colors during the color propagation phase: (a) producing a color slightly different from the existing ones and (b) giving a neutral grey color, we propose (a) changing the cyclic traversal order of colors and (b) assigning to a cluster a color that is non-complementary with respect to the neighbor clusters.

Figures 1(a), 8(b), 11(a), and 12(a) illustrate the perceptual intuitiveness of our color propagation scheme when moving throughout the sequence. The same algorithm can also be used for any segmentation pair without the need of history context. Thus, we have applied this approach to demonstrate the color transition results between (a) the variable segments and (b) the variable resolution segments obtained by our method. Finally, Table 1 shows its interactive nature when rendering several clustering sequences (from the high-detail elephant gallop (188 fps) to the low-detail hand animation (715 fps)).

4.2. Real-time Segmentation

Our framework can efficiently handle segmentations of streamed or dynamically created mesh animations without the need of downloading the pre-processed animation frames for off-line segmentation. This is a key feature that is not offered by previous approaches. The idea is to merge the newly "arrived" pose partitioning with the segmentation resulted from joining the partitionings of all previous poses (see Figure 3(b)).

Figures 8(a) and 11(b),(c) provide thorough examples of incrementally merging a number of pose partitionings to generate the final segmentation of real-time mesh animations. Observe that cluster-refinement is omitted at each frame in Figure 8(a). On the other hand, the intermediate segmentations are enhanced by the cleaning procedure in Figure 11(c).

4.3. Variable Segmentation

We consider the problem of clustering data over time, maintaining simultaneously two conflicting criteria: (a) remain faithful to the past-data and (b) effectively alter when moving on to future data. This application is helpful to detect *motion changes* at each time step, contrary to the information extracted by traditional segmentation methods desired for subsequent shape analysis and geometry processing applications.

Despite clustering independence between individual successive poses, we offer users with a smooth transition between pose-to-pose clusterings, by joining the optimal partitioning of the current pose with that of the next pose (see Figure 3(c)). Due to the small number of resulting clusters, cleaning can be avoided. Our method is highly efficient when compared to previous variable segmentation methods (evolutionary versions of classic clustering methods [CKT06] and splitting/merging operation on the past-clustering [ACH*13]) and can achieve interactive performance when a fast per-pose clustering is employed (for computation times see Table 1). Figures 8(c) and 12(b) include variable segmentations of a cloth simulation and a dance animation, respectively. Observe that the temporal consistency is better preserved when the joined clustering sequence is used as compared to the individual pose clusterings.

4.4. Multi-resolution Segmentation

Contrary to bottom-up and top-down hierarchical clustering methods which can only merge or only split clusters to reach the desired segmentation solution, we provide users with an interactive tool to adapt resolution of the final segmentation. As discussed above, our approach aims at cleaning small-area clusters which usually correspond to highly-deformable regions. Starting from a noisy over-segmentation, users can efficiently simplify it by adjusting the h parameter. Intuitively, the more h is increased, the larger the parts to be removed. However, they are upper-bounded by the resolution of the initial over-segmentation.

Figures 1(c) and 12(c) illustrate how the rigidity in the segmentation is preserved when the level of detail is decreasing. Table 1 shows the performance efficiency of the p2p-cleaning process for various mesh animations. Note that this process does not depend on mesh geometry size.

4.5. Combine Segmentations of Mesh Animations

Except from joining pose partitionings to derive a final segmentation, our framework can easily combine different segmentations of one or more mesh animations. Note that our approach can only work when a bijective vertex mapping between the animations has been established.

Figure 9 illustrates the *segmentation transfer* output between individual mesh animations. This is very helpful since animators may avoid the burdensome manual work of producing the intermediate result. The global segmentation of each animation produces a better partitioning of each movement leading at a superior merged segmentation when compared to the one derived from joining both animations. This is due to the reduced multi-modal distribution of the feature space computed from the motion of the merged animation.

4.6. Modifying Mesh Animations

Using our framework, we can avoid the segmentation re-computations when the user performs editing or extending operations on the original animation sequence [CH12]. Since the over-segmentation result does not depend on the joining order, we can simply join the new partitioning (from the edited or the added pose) with the final segmentation of the original animation.

Figure 10 illustrates how a segmentation, computed from merging clusterings of an initial set of flamingo poses, is adjusted to reflect the motion of two newly added poses. On the other hand, Figure 11(c) demonstrates how the final segmentation of a mesh animation is efficiently altered when pose editing is performed. For clarity, we provide

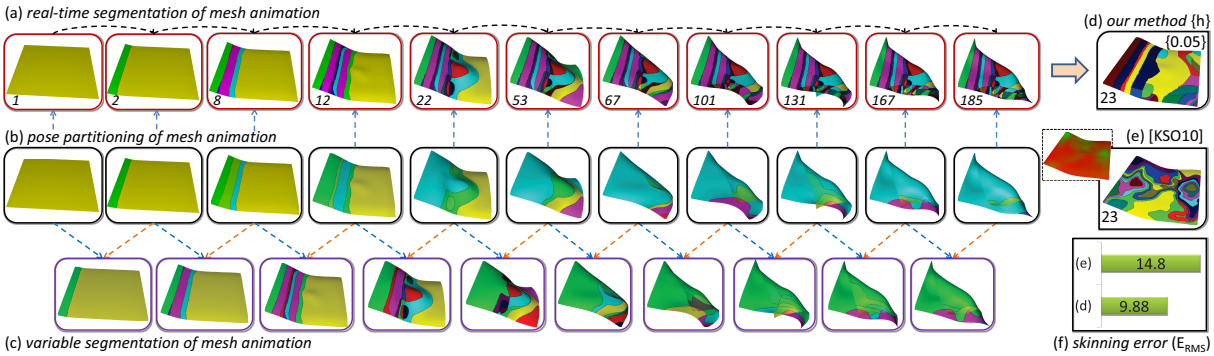

Figure 8: (a) Real-time segmentation construction process. (b) Pose partitioning enhanced by our color propagation scheme. (c) Variable segmentation. (d) Our output is superior in the context of (f) skinning error when compared to (e) the one of [KSO10].

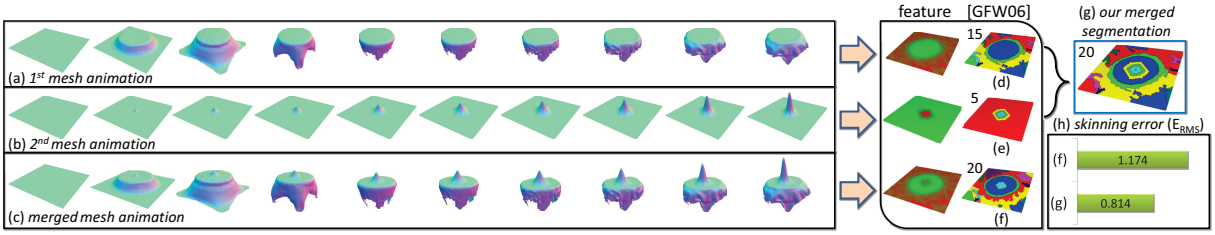

Figure 9: Merging (d),(e) the global segmentations from [GFW*06] of (a),(b) two individual mesh animations. (h) We observe the skinning error superiority of (g) our merged segmentation when compared with (f) the global segmentation [GFW*06] of (c) the animation created by merging both animations.

the intermediate steps of the incremental merging strategy despite the fact that the same segmentation can be produced by merging the partitioning of the edited poses with the final segmentation of the original animation.

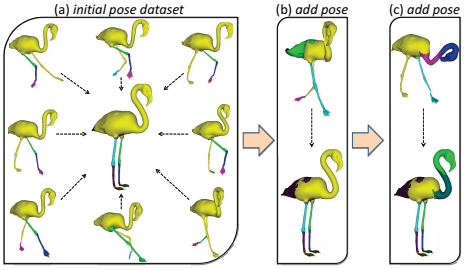

Figure 10: (a) Final segmentation constructed by joining 8 initial pose partitionings. The segmentation is refined after adding (b) initially a new pose, (c) followed by a second one.

5. Experimental Study

We evaluate our proposed segmentation technique with respect to performance and quality under a broad set of testing inputs. These include rigid, highly-deformable and hybrid mesh animations. Table 1 summarizes the geometry properties and clustering details for each animation. For more segmentation results, we refer readers to watch the accompanying video. The experiments were performed on an Intel Core i7 870 @2.83GHz CPU using multi-threaded implementation.

A variation of a top-down hierarchical clustering technique [GFW*06] is used in our experiments for primary pose-decomposition. Rotation angles, extracted from the deformation gradients [SP04] computed with respect to the rest-pose, define the one-dimensional feature space. We compare our segmentation results with the ones derived by a variety of widely-accepted global segmentation methods using the same number of desired segments. This is accomplished by accurately adjusting the value of *h*. Without loss of generality, we have used uniform seeding and the same number of iterations (5) for all clustering algorithms and 1% of the vertices are used as initial input for spectral clustering [DATTS08]. Finally, K-means [LD12] and spectral clustering [DATTS08] may result in segments with several disconnected components when segmenting non-rigid animations making these results unable to support several graphics applications.

5.1. Performance Analysis

Table 1 presents a comparative performance overview of the intermediate steps employed by our framework to produce a final segmentation. The computation times for all steps exhibit a linear behavior on the mesh geometry size, which is consistent with our time complexity analysis. Furthermore, note how the over-segmentation and cleaning performance scales linearly when the number of per-pose

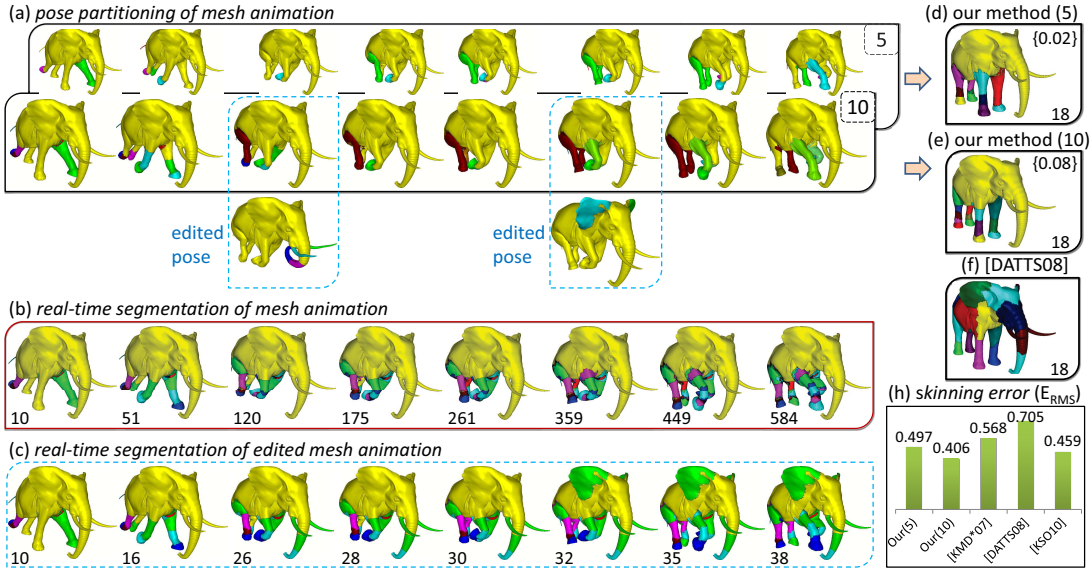

Figure 11: (a) Smooth visual transitions of the pose partitionings with 5 and 10 components. Editing operations are highlighted from the partitioning of the modified poses. Intermediate real-time segmentation steps (b) before and (c) after editing is applied. (d),(e) Contrary to our refined segmentations, (f) output of [DATTS08] results at wrongly decomposing non-animated areas.

segments increases at the elephant mesh animation. The efficiency of our framework is constrained by the individual pose decompositions which take more that 90% of the total computation time. Finally, note that we cannot support interactive performance for segmenting mesh animations when the top-down hierarchical clustering is used. Moving to multi-source region growing [KSO10] as initial partitioning, we achieve 6 fps when real-time segmenting the hand animation (Figure 1). A GPU-accelerated clustering may be explored as an alternative to speed up performance.

Figure 12(f) shows that our algorithm is better in terms of performance when compared to a variety of global segmentations. This is due to the fact that segmentations that explore spectral clustering [DATTS08] or skinning transform matrices [GFW*06, SY07, LD12] as motion characteristic suffer from high computation times. Mean-shift clustering aware methods [JT05, LXL*12] should also be avoided to produce fast global segmentations. On the other hand, region-growing [KSO10] is faster than our method in low resolutions. However, this comes with the price of limited quality of the generated partitions. Contrary to prior art, our method is only slightly affected when changing from one segmentation resolution to another one.

5.2. Quality Analysis

Contrary to static meshes where several definitions and metrics have been introduced to define optimal segmentation depending on the application objective [CGF09], a framework for the objective evaluation of segmenting mesh animations is missing. Intuitively, when mesh movement is defined as a function of an underlying skeleton, the segmentation objective is to partition the surface into meaningful volumetric parts. On the other hand, segmentation of a highly-deformable object is targeted at decomposing the mesh into surface patches with similar motion characteristics.

In this work, we evaluate our method in a *skinning context*: how well the segmentation-aware compressed animation reproduces the original animation when linear blend skinning is used: A simple assignment method [KMD*07] is initially employed to set the skinning weights $w_{i,j} \in [0,1]$ which describe the amount of influence of j-th segment on i-th vertex. A fitting process is followed by computing the matrices $M_j^t \in \mathbb{R}^{3\times4}$ that describe the transformation of each segment j from the rest-pose p_0 to the subsequent poses $p_t, \forall t \in [1,k]$. The transition of vertex i from the rest-pose v_i^0 to a pose p_t is then described by $\hat{v}_i^t = \sum_{j=1}^{S} w_{i,j} M_j^t v_i^0$, where S is the total number of segments. The E_{RMS} error metric proposed by [KSO10] is used to measure the mean skinning approximation error of the animation sequence. More specifically, $E_{RMS} = 1000 \frac{\|A-A'\|_F}{3nk}$, where A and A' are $3k \times n$ matrices that contain the original ($\{v_i^t\}$) and skinned approximated ($\{\hat{v}_i^t\}$) coordinates of each vertex throughout the animation sequence, respectively.

Rigid Animations. Figure 1(e) shows the comparison of our method in terms of extracting rigid parts when segmenting an animated hand. Low-resolution global segmentations [SY07] fail to accurately partition most of the articulations (e.g middle finger). These segments are captured at higher-detail representation with the burden of noise cluster creation. From the E_{RMS} table, we observe that the behavior of our method starts to change when the number of the final segmentation resolution is low. This is reasonable since high-resolution segmentations consist of numerous tiny noisy clusters generated between consecutive partitionings. Increasing the

number of per-pose extracted segments will enhance the skinning approximation.

In Figure 11, pose partitionings of 5 and 10 components are merged to construct multi-resolution segmentations of an elephant gallop animation. First, we observe that the quality of the former is insufficient due to the low number of segments per pose. This results in a significant loss of semantic part information such as the knee of the front-left foot (red-painted). The segmentation quality is sufficiently improved when more per-pose clusters are used. The accompanying table illustrates the skinning superiority of our method when compared with several methods on a 18-component segmentation.

Deformable Animations. Figures 8 and 9 describe objects that deform under no skeletal influence. Figure 8 shows a segmentation that consists of 23 components from a cloth simulation. Note that the number of per-pose clusters is not-constant. Our segmentation preserves better spatial coherency without creating irregular shapes when compared to the global segmentation [KSO10] extracted from the illustrated feature space.

A representative example of combining individual segmentations extracted from tablecloth mesh animations is shown in Figure 9. The segmentation of the first mesh animation is efficiently transferred to the second one. Mean rotation angle was used to define the feature space. Note that the merged segmentation is superior when compared to the global segmentation [GFW*06] of the animation created by blending both animations (e.g the highly-animated protrusion region was captured by only one segment).

Hybrid Animations. Highly deformable objects can be used to model clothes in conjunction with skeletal animation. Figure 12 illustrates how our approach produces segmentations that accurately partition the rigid parts (head, arms and legs of the dancer) from the highly-deformed surfaces (the dress follows the motion of the dancer) of a samba dancing animation. On the other hand, global segmentation [SY07] produces low-quality partitions. Despite the sufficient rigidity captured at low-detail, the right leg is wrongly connected to the dress. Moving to higher resolutions, we observe that rigid components (head and legs) are significantly being "pruned" creating meaningless parts. This leads to a decreasing consistency of the overall segmentation. Similarly to the hand animation, we observe that moving from a high-to-low dimension our method behaves better as compared to the rest methods when skinning is used to approximate the initial mesh animation.

5.2.1. Discussion

We generally notice that our algorithm produces better segmentations when focusing on the visual quality of the segmentations for the mesh animation, whether partitions it into meaningful volumetric or surface parts. A centric observation is that it is better to combine multiple partitionings and simplify them, rather than trying to attempt a global segmentation directly. However, the final output depends on the quality and the number of the individual decompositions extracted from each pose. We can explore a number of directions to improve pose clustering quality by (a) employing a more sophisticated clustering algorithm, (b) rectifying the generated boundaries or (c) imposing a confidence criterion on the initial pose partitionings to retain only important boundaries. However, this will come with the burden of additional parameter tuning and extra computation cost. Note that an efficient low-detail segmentation can only be computed in case where the individual pose partitionings exhibit high similarity. Furthermore, our method is suitable for computing accurate high-resolution segmentations when the number of per-pose segments is maintained at increased levels. Conversely, merging partitionings that do not capture the desired information, would normally lead to poor final segmentation (e.g. quality of Figure 11(d)).

6. Conclusion

We have presented a general approach to efficiently deriving a multi-resolution segmentation of arbitrary deformations based on building an over-segmentation which can optionally be simplified by a robust cleaning reduction process. The final segmentation can accurately be adjusted when the original mesh animation sequence is either modified or updated. A smooth frame-frame clustering transition is offered by merging the partitionings between consecutive poses. The resulting segments are painted on the fly by a novel color propagation scheme. Finally, we have included extensive comparative results with respect to performance and quality.

Limitations/Future Work. There is a number of research directions that could be explored further to improve the limitations of the current work. First, our approach may easily support time-varying meshes [ACH*13] by exploiting vertex mapping techniques to establish pairwise parameterization between successive frames. Another challenge is to cope with interactive segmentation on large meshes by taking advantage of the GPU high-performance parallel architecture. Further directions may be investigated for tackling the problem of the increased resolution of over-segmentation. For example, we may reorganize similar poses into clusters and pick one of them thereby reducing significantly the number of the merged partitionings. Except from improving performance, this solution will also reduce the additional memory requirements of storing the individual per-pose partitionings. Finally, visualization coherency is lost between "far-away" poses (e.g. observe the different assigned colors at the head segment at the fifth and seventh pose in Figure 12 (a)). This happens when there is no mapping between clusters of successive poses. An interesting alternative strategy could be to increase the cluster mapping search in a larger time window.

Acknowledgements. This research has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program Education and Lifelong Learning of the National Strategic Reference Framework (NSRF) - Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.

References

[ACH*13] ARCILA R., CAGNIART C., HÉTROY F., BOYER E., DUPONT F.: Segmentation of temporal mesh sequences into rigidly moving components. *Graphical Models 75*, 1 (2013), 10–22. 2, 6, 9

[AS07] AMJOUN R., STRASSER W.: Efficient compression of 3d dynamic mesh sequences. *Journal of WSCG 15*, 1-3 (2007). 1, 2

				Per pose						Mesh Animation			
					Feature	Clustering		Variable Segmentation		Over-segmentation	p2p-Cleaning	Total	
Mesh Animation	Vertices	Faces	Poses	Clusters	Compute	Compute	Propagate Colors	Compute	Propagate Colors	Compute (segments)	Compute (segments)	real-time	off-line
Hand	7929	15855	22	12	0.116	1.603	0.0014	0.0046	0.00084	0.215 (379)	0.03 (24)	1.72	37.95
Elephant Gallop	42321	84638	24	5	0.483	3.327	0.0053	0.012	0.0041	0.735 (743)	0.07 (18)	3.844	92.25
				10	0.483	6.463	0.0053	0.012	0.0041	1.475 (1729)	0.09 (18)	7.281	174.7
Flowing Cloth	25921	51200	19	[2,5]	0.295	1.873	0.0028	0.0074	0.0025	0.698 (400)	0.02 (23)	2.205	41.91
Samba	9971	19938	24	5	0.158	0.734	0.0013	0.024	0.0009	0.78 (1016)	0.07 (12)	0.927	22.26

Table 1: Extensive performance comparison (in seconds) of the algorithmic steps of our method to segment various mesh animations.

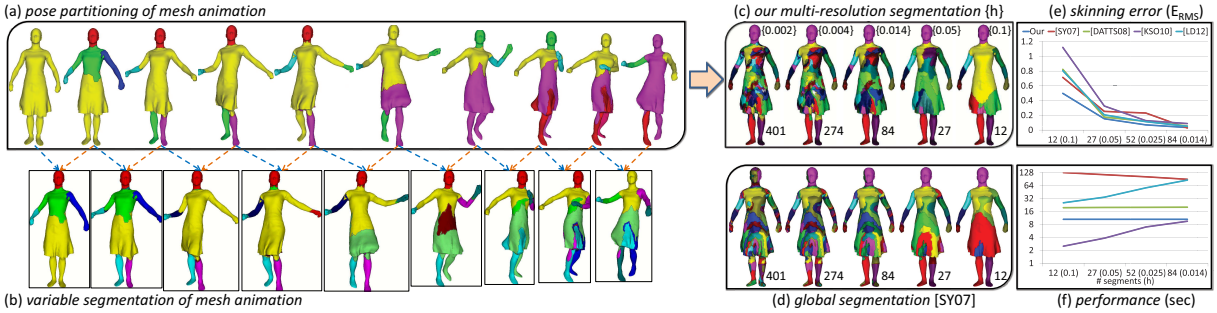

Figure 12: (a) Propagating segment colors through the sequence. (b) Variable segmentation. (c) Our multi-resolution segmentations accurately divide rigid parts from non-rigid surfaces even in high resolutions as opposed to (d) the ones of [SY07].

[CGF09] CHEN X., GOLOVINSKIY A., FUNKHOUSER T.: A benchmark for 3d mesh segmentation. *ACM Transactions on Graphics (TOG) 28*, 3 (2009), 73. 8

[CH12] CASHMAN T. J., HORMANN K.: A continuous, editable representation for deforming mesh sequences with separate signals for time, pose and shape. *Computer Graphics Forum 31*, 2pt4 (2012), 735–744. 2, 6

[CKT06] CHAKRABARTI D., KUMAR R., TOMKINS A.: Evolutionary clustering. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining* (2006), ACM, pp. 554–560. 2, 6

[DATTS08] DE AGUIAR E., THEOBALT C., THRUN S., SEIDEL H.-P.: Automatic conversion of mesh animations into skeleton-based animations. *Computer Graphics Forum 27*, 2 (2008), 389–397. 1, 2, 7, 8

[DGGV08] DE GOES F., GOLDENSTEIN S., VELHO L.: A hierarchical segmentation of articulated bodies. *Computer Graphics Forum 27*, 5 (2008), 1349–1356. 2

[DIHF12] DU P., IP H. H.-S., HUA B., FENG J.: Using surface variability characteristics for segmentation of deformable 3d objects with application to piecewise statistical deformable model. *The Visual Computer 28*, 5 (2012), 493–509. 2

[FKY*10] FENG W.-W., KIM B.-U., YU Y., PENG L., HART J.: Feature-preserving triangular geometry images for level-of-detail representation of static and skinned meshes. *ACM Transactions on Graphics (TOG) 29*, 2 (2010), 11. 2

[GFW*06] GÜNTHER J., FRIEDRICH H., WALD I., SEIDEL H.-P., SLUSALLEK P.: Ray tracing animated scenes using motion decomposition. *Computer Graphics Forum 25*, 3 (2006), 517–525. 1, 2, 7, 8, 9

[HFL12] HU R., FAN L., LIU L.: Co-segmentation of 3d shapes via subspace clustering. *Computer Graphics Forum 31*, 5 (2012), 1703–1713. 3

[HTRS10] HASLER N., THORMÄHLEN T., ROSENHAHN B., SEIDEL H.-P.: Learning skeletons for shape and pose. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games* (2010), ACM, pp. 23–30. 1, 2

[JKS05] JULIUS D., KRAEVOY V., SHEFFER A.: D-charts: Quasi-developable mesh segmentation. *Computer Graphics Forum 24*, 3 (2005), 581–590. 2

[JT05] JAMES D. L., TWIGG C. D.: Skinning mesh animations. *ACM Transactions on Graphics 24*, 3 (2005), 399–407. 1, 2, 8

[KMD*07] KAVAN L., MCDONNELL R., DOBBYN S., ŽÁRA J., O'SULLIVAN C.: Skinning arbitrary deformations. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games* (2007), ACM, pp. 53–60. 2, 8

[KSO10] KAVAN L., SLOAN P.-P., O'SULLIVAN C.: Fast and efficient skinning of animated meshes. *Computer Graphics Forum 29*, 2 (2010), 327–336. 1, 2, 7, 8, 9

[LD12] LE B. H., DENG Z.: Smooth skinning decomposition with rigid bones. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 199. 1, 2, 7, 8

[LWC06] LEE T.-Y., WANG Y.-S., CHEN T.-G.: Segmenting a deforming mesh into near-rigid components. *The Visual Computer 22*, 9-11 (2006), 729–739. 1, 2

[LXL*12] LIAO B., XIAO C., LIU M., DONG Z., PENG Q.: Fast hierarchical animated object decomposition using approximately invariant signature. *Visual Computer 28*, 4 (2012), 387–399. 2, 8

[Sha08] SHAMIR A.: A survey on mesh segmentation techniques. *Computer graphics forum 27*, 6 (2008), 1539–1556. 1, 2

[SP04] SUMNER R. W., POPOVIĆ J.: Deformation transfer for triangle meshes. *ACM Transactions on Graphics (TOG) 23*, 3 (2004), 399–405. 2, 7

[SY07] SCHAEFER S., YUKSEL C.: Example-based skeleton extraction. In *Proceedings of the fifth Eurographics symposium on Geometry processing* (2007), pp. 153–162. 1, 2, 8, 9, 10

[WB10] WUHRER S., BRUNTON A.: Segmenting animated objects into near-rigid components. *The Visual Computer 26*, 2 (2010), 147–155. 3

[WPP07] WANG R. Y., PULLI K., POPOVIĆ J.: Real-time enveloping with rotational regression. *ACM Transactions on Graphics (TOG) 26*, 3 (2007), 73. 2