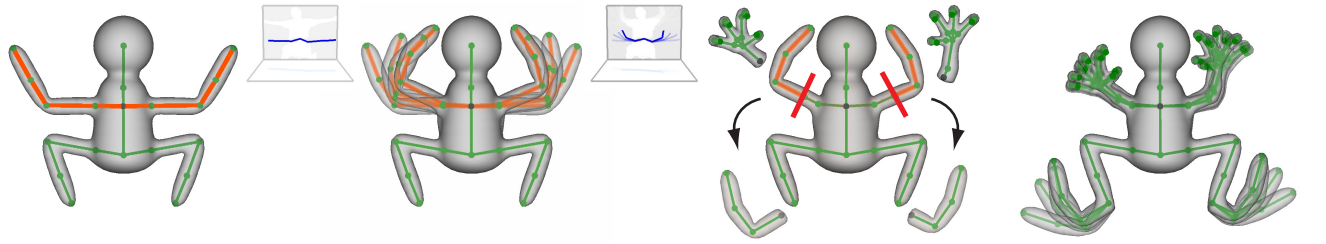
AniMesh: Interleaved Animation, Modeling, and Editing

Ming Jin[1] Dan Gopstein[1] Yotam Gingold[2] Andrew Nealen[1]
[1]New York University [2]George Mason University



(a) *Pose (inset) and select parts of the target shape (orange)* **(b)** *Animation transfer* **(c)** *Shape editing* **(d)** *Animation playback after editing*

Figure 1: *The user poses to select parts of a model for animation (a). Thereafter, the user motion is applied to the matched portion of mesh (b). She is free to interleave editing operations (c), and our system automatically adapts recorded animations to the updated model (d).*

Abstract

We introduce AniMesh, a system that supports interleaved modeling and animation creation and editing. AniMesh is suitable for rapid prototyping and easily accessible to non-experts. Source animations can be obtained from commodity motion capture devices or by adapting canned motion sequences. We propose skeleton abstraction and motion retargeting algorithms for finding correspondences and transferring motion between skeletons, or portions of skeletons, with varied topology. Motion can be copied-and-pasted between kinematic chains with different skeletal topologies, and entire model parts can be cut and reattached, while always retaining plausible, composite animations.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

Keywords: Motion transfer, motion editing, animation interfaces, skeletal animation, shape editing, mesh modeling

1 Introduction

The character animation pipeline typically requires creation of detailed 3D characters, high quality rigs—skeletal bones and skin weights binding the surface to them—and lively motion sequences. Each stage in this sequence is time-consuming and requires task-specific proficiency, making the process inaccessible to novices. Furthermore, this sequence is linear. A model cannot be edited once it has been rigged and animated, impeding iterative design. Motivated by this observation, we created AniMesh, a system that unifies modeling and animation. Surfaces can be refined, sculpted or assembled; skeleton and skin weights can be re-designed and re-painted; and animations can be quickly prototyped and verified. Our design goal, unlike most existing animation suites, is to support tasks traditionally poorly served by modern tools (e.g. rapid prototyping, accessibility to non-expert users, exploration), where a non-linear and non-destructive workflow can offer advantages over more traditional techniques.

Prior work has addressed similar issues by automating the modeling process [Igarashi et al. 1999; Nealen et al. 2007], enabling the reuse of rigs [Baran and Popović 2007; Miller et al. 2010], or integrating modeling and rigging into one system [Borosán et al. 2012]. However, a simple operation such as removing part of a limb from an animated character and re-attaching it elsewhere would still invalidate the animation, requiring it be re-designed. If we view the modified character as an entirely different shape, then this problem can be reduced to motion retargeting [Gleicher 1998]. Retargeting, however, is typically treated as a stand-alone process that relies on similar topology between the source and target, or manually specified mappings. Furthermore, structural changes applied to the shape force the retargeting process to be reapplied.

We incorporate motion retargeting into a unified process for designing shapes *and* their animations. During animation design and retargeting, our system provides users with guided, direct control (Figure 1a and b). For editing operations, instead of globally re-establishing the correspondence after any edits, our retargeting method locally maintains the originally mapped motions on every single skeletal bone (Figure 1c and d). With these two supporting components, we provide a non-destructive animation creation and editing tool that allows users to change the shape, the underlying skeleton, and the animation in any order, using the basic operations of cutting and merging, as well as skeletal node addition and deletion. Specifically, our contributions provide:

- a non-linear modeling/animation tool easily accessible to novice users and suitable for rapid prototyping
- and a shape editing (cutting, merging, node addition and removal) procedure that maintains consistent modeling and animation transformations.

To facilitate the creation and editing of animations in AniMesh, we have developed a partial skeleton matching algorithm based on shape abstraction, as well as introduce a motion representation and mapping algorithm that allows for multiple reuses of existing source motions. For the purposes of this system we ignore aspects in retargeting such as environmental interactions and only focus on the transfer of motion from one skeleton to another. We refer to an individual animation as a sequence of rotations over time due to motion of the source skeleton. The source animation is record-

ed and stored, and can later be sampled for the purpose of retargeting onto target kinematic chains with different degrees of freedom (DOFs). On the target skeleton, for each bone we store references to the source animation(s) as well as its own modeling transformation. This representation is essential for integrating animation retargeting into the modeling operations of posing, cutting and merging (Section 6). Our system is built on the shape, skeleton, and mesh/skin editing system RigMesh [Borosán et al. 2012], and while we do not describe that system in detail, we use their local mesh adjustment and reskinning algorithms in this paper.

We see AniMesh as a step towards non-linear, non-destructive 3D modeling, editing, rigging, and animation. All components and algorithms are designed to allow a single user to interleave these tasks and significantly decrease iteration time, and, ideally, increase their prototyping abilities. Due to this design goal, certain classes of animations are outside the scope of this work. For example, adding root translation introduces issues like footskate. Similarly, since our system is based on skeletal animation, we do not address facial animations and other advanced features using non-skeletal rigs.

2 Related Work

Any motion retargeting approach requires, at its core, a mapping between the source and the target. Most of the existing work in this field [Popović and Witkin 1999; Lee and Shin 1999; Shin et al. 2001; Sumner and Popović 2004; Kulpa et al. 2005] address the issue of retargeting to human-like figures or require that the source and the target figure have similar topologies. Gleicher [1998] describes retargeting to a figure with different DOFs, under the constraint that the characters have approximately the same size. Dontcheva et al. [2003] propose to layer animations designed at different passes, effectively retargeting the source animation from known widgets onto characters with vastly different shapes. Hecker et al. [2008] describe a system used in the game SPORE that retargets pre-authored animations to player-generated characters whose skeletal topologies are unknown when designing the animations. They require animators to provide semantic information on motions they design for a general reference character. Using a statistical mapping function, Yamane et al. [2010] map human motion capture data to a non-humanoid character. Poirier and Paquette [2009] use multi-resolution subgraphs and a shape descriptor for partial matching and retargeting. Their retargeting method has quadratic runtime complexity w.r.t. the number of embedding points, whereas our method runs in real time. Miller et al. [2010] retarget skinning information using a database of partial rigs – skeleton and skin weights identifying the surface of the model with skeletal bones. In comparison, our method does not require such examples and works directly with the user input.

Many existing motion retargeting approaches [Gleicher 1998; Monzani et al. 2000; Hecker et al. 2008] use inverse kinematics (IK) [Watt and Watt 1991] to determine the rotation for each target bone, preserving end effector positions. In our setting, IK could lead to unnatural or even impossible solutions, since bone lengths differ between the user and model. It is more important to generate plausible motion transfer. This has been addressed in the literature by the work of Mine et al. [1997]. Informed by proprioception, Boff et al. [1986] provide body-relative feedback for the user. This inspired us to design a simple and *predictable* retargeting method. Predictability is a key benefit for animators who must mentally invert the mapping in order to achieve desired motions. Our motion representation and sampling method (Section 5.1) allows the motions to be carried over during modeling, incorporating motion retargeting in the shape and model editing process.

With advances in inexpensive motion sensors, more recent works have approached motion retargeting using devices such as Kinect [Microsoft 2015] and LeapMotion [2015]. Chen et al. [2012] allow the user to pose the input skeleton and embed it within the 3D scanned static mesh using the Kinect. Extending the embedded deformation framework [Sumner et al. 2007], they solve for the optimal transformation on the deformation graph. Vögele et al. [2012] map the Kinect skeleton of two humans to a quadrupedal target skeleton using a part-based statistical shape model. Seol et al. [2013] show a motion puppetry system that drives a non-humanoid with fewer DOFs than those of the performed source animation. Rhodin et al. [2014] allow for puppeteering using arbitrary source motions. Their source and target are not necessarily skeletal motions, and the user needs to define a small number of pose correspondences between source and target motions. Using an automatic method to embed the skeleton inside the shape, Jacobson et al. [2014] introduce an iterative system for manipulating virtual characters using their own skeletal animation input device. In AniMesh, the user also has direct control over the matching process by posing and selecting the desired skeletal matches. Our method supports various input motion data, and works within the skeletal deformation framework.

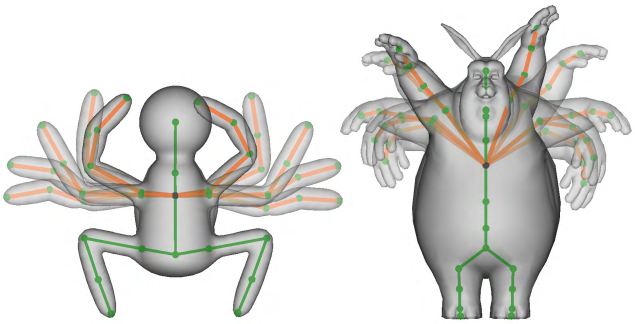


Figure 2: *Manually specifying the mapping and the transfer. The user selects upper body skeletal chains from the source shape (left), specifies its corresponding chain on the target (right), and transfers the motions. The matched skeletal chains are highlighted.*

3 Workflow

An example workflow using AniMesh is as follows. The user begins by loading a rigged mesh (the *target*) and uses a *source* of live bone-skeleton motion data. Our system is data-source agnostic, and supports motion data provided by Microsoft Kinect (the humanoid shadow and skeleton of the arms in Figure 1a), LeapMotion, or a pre-existing motion library. By orienting the target shape on-screen and then posing in front of the capture device, a portion of the model's skeleton is selected for animation. Our matching algorithm (Section 4) finds a set of matches between the user's source skeleton and the target skeleton; the top match is displayed in real-time (Figure 1a). The user cycles through and selects one of a number of viable candidate matches, assumes a comfortable pose (the rest state), and initiates simultaneous motion capture and animation transfer (Figure 1b and Section 5). After recording, the user may play back or scrub through the recorded animations on a timeline, or edit the model by cutting, copying, and merging; she can also re-use existing models and parts (Figure 1c). Animations can be layered by recording overlapping motion or by re-using performances through the basic operations of copying/pasting (of shapes as well as animations), cutting and merging. User may also manually override the automatic correspondence of skeletal chains between the source and the target shapes (Figure 2). All operations are demonstrated in the accompanying video.

The key to our non-destructive shape modeling/animation workflow is the ability to maintain motions while editing the shape (Section 6). This relies on our approach to help the user select a skeleton correspondence (Section 4) and our approach to efficiently represent and transfer motion (Section 5).

4 Skeleton matching by shape abstraction

To select a portion of the model for animation, as well as find a suitable skeletal chain for live motion retargeting and visual animation feedback, we find and score candidate partial matches between the source and target at various scales. Our matching process considers both skeletons' topology and geometry.

Topology Animators have the freedom to map their (full or partial) source skeleton to any part of the target. In our system, the source can be the user's input motion data or some existing animation attached to a skeleton, and the target shape can be highly varied in its skeletal topology. Both source and target graphs are limited only in that they are trees, i.e. connected and acyclic.

Geometry Animators pose and orient parts of their bodies in space to select an intended match between the source and target; the matching, therefore, should be dependent on spatial similarity, and also be scale-invariant. Finding a good similarity metric not only affects the quality of the mapping, but also influences the predictability of its presentation to the animator.

In general, skeleton matching is a subset of the well-studied topics of shape correspondence and graph matching. For a survey of modern methods see [Van Kaick et al. 2011]. In our system, the user specifies the semantics of a mapping by changing the pose in front of the capture device. Thus, we must take into account both mathematical and perceptual aspects of the problem. Our matching algorithm uses concepts from co-abstraction [Yumer and Kara 2012] to simplify the trees, and concepts from combinatorial tree search to select them. We apply an abstraction process to the source and the target skeletons independent of each other. This results in a level-of-detail (LOD) hierarchy for the target skeleton, and a single-step simplification of the source (Section 4.1). Afterwards we collect all candidate sub-trees in the target LOD hierarchy and compare them with the simplified source to rank them for suitability as a source-target mapping (Section 4.2). This process is illustrated in Figure 3.

4.1 Topology: hierarchical skeleton abstraction

In general, a one-to-one mapping between a puppeteer and a model may not exist. Instead of always mapping the entire source skeleton from the capture device to all parts of the target shape simultaneously, the animator is better served by being able to animate subsets of the target in different passes. In AniMesh, we use a LOD representation to allow the animator to choose whether to control the entire model broadly, or to select and animate small details individually.

In our abstraction process, we categorize skeletal nodes as either *branch* nodes (more than two neighbors). For a target skeleton with known root node, collecting candidate matches could proceed by traversing the tree and selecting every sub-tree from the root down to each leaf. Instead, as a root node is not always given–and we prefer that novice users need not worry about this technical detail–we reverse this process and, starting from the leaf nodes, create the LOD hierarchy, simplify the sub-trees for subsequent matching, and find a suitable root node, all in tandem.

To illustrate our hierarchical matching algorithm, we use the target skeleton in Figure 3c. **Step 1.** Each iteration begins by identifying leaf nodes, shown as red circles. According to our topology criteria, the chain nodes (blue circles) between a leaf and a branch node (orange circle) form a trivial path, and are thus removed. Their geometric positions, however, should still contribute to the shape of the skeletal chain for computing the similarity (Section 4.2). **Step 2.** We update the leaf positions to be the average of deleted chain nodes, weighted by their outgoing bone lengths (Figure 3d). In addition, the lengths of the removed chains are accumulated and stored at the leaf nodes, to be used for further abstractions (Step 4). **Step 3.** Sub-trees are generated by identifying their roots at branch nodes with *only one* non-leaf neighbor (green circles). In Figure 3d, the candidate sub-trees identified in the first iteration (blocks 1-1 and 1-2) are collected for matching. It is important to note our criteria for identifying sub-trees: in Figure 3d, branch nodes p and q each have two non-leaf neighbors. They will be identified as sub-tree roots at a later iteration (Figure 3e). **Step 4.** As a last step in the iteration, each sub-tree is abstracted into a leaf node for subsequent iterations (e.g. blocks 1-1 and 1-2 are turned into leaves in Figure 3e). Its geometric position is determined as the average of its nodes' positions, weighted by their accumulated chain lengths.

This iterative process terminates once we have found the root node of the hierarchy. If a single node remains, it is declared to be the global root. If a chain remains, then the node (black circle in Figure 3f) closest to the center of mass is chosen. The animator also has the option to manually specify the root if necessary, though we have not found this necessary in our experiments.

4.2 Geometry: match ranking

Once all candidate sub-trees are identified and collected (Figure 3g, left), we compare them to the simplified source tree (Figure 3b), thus providing potential matches over all levels-of-detail on the target. For each candidate match, the m branches of the source ($m = 2$ in Fig. 3b) are compared to the n branches of the candidate sub-tree, and a matching score is calculated for each of the $\binom{n}{m}$ possible matches per sub-tree; a global ranking of these partial

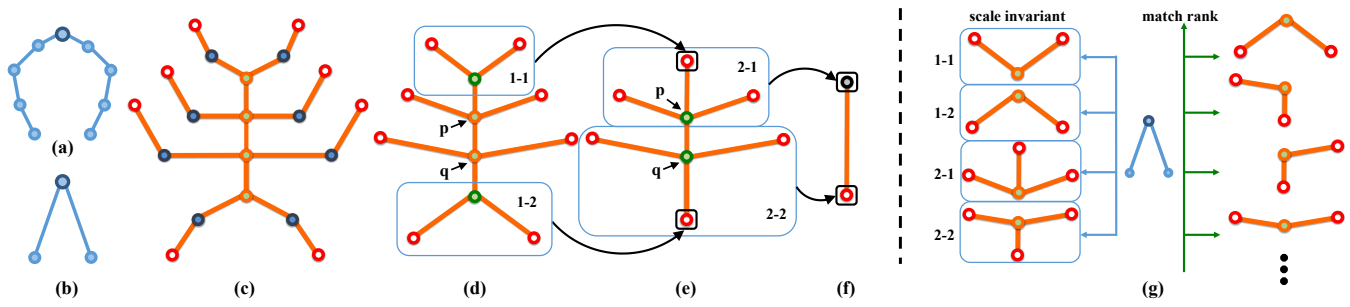


Figure 3: *Finding and scoring candidate partial matches between the source and target skeletons on various scales. We use the upper body skeleton from Kinect (a) as the source, with its abstraction in (b). The LOD hierarchy of the target skeleton (c) is identified iteratively (d)–(f). (g, left) Our algorithm collects sub-trees (1-1, 1-2, 2-1 and 2-2) at each iteration, and uses them to find the candidate mapping with the simplified source (b) in a scale-invariant manner. Mappings are ranked using our proposed similarity metric (g, right).*

We made the design choice to compute the resemblance in 2D (screen space), similar to looking in a mirror. We furthermore consider resemblance between source and target to be translation and scale invariant, thus, sub-trees are translated such that their root positions coincide, and uniformly scaled such that each of their longest branches has length one. Scale invariance is of particular importance in our modeling and animation pipeline: because the animator can disassemble, scale, and re-attach parts of the original shape, the target shape may contain similar or identical features at different scales. However, the animator cannot modify the scale of the input skeleton (e.g. the length of her arm) other than physically changing the distance to the capture device or zooming the viewing camera.

Our similarity metric is defined as the sum of cosine similarities (i.e. sum of dot products) between the m pairs of branches on the abstracted source and target sub-trees. For each of the $\binom{n}{m}$ possible matches, its matching score is chosen as the highest similarity among the $m!$ possible pair (source and target) permutations. This can easily be computed in real-time as m is generally very small (for the Kinect input with two arms, $m = 2$). This metric preserves rotational variance, and matches the orientation of the user input as closely as possible. Figure 3g shows the result of comparing the abstracted source tree against the identified target sub-trees. A total of 16 candidate matches are scored and ranked, and the top 4 matches are visualized.

5 Motion representation and retargeting

In our system, source motions can originate from various capture devices or be reused from other models. Given a correspondence between source and target skeleton abstractions, as well as a user-provided source animation, our next task is to transfer the animations between their corresponding polylines. As we cannot make any assumptions about the sampling of the target polyline, we propose a motion representation independent of its eventual target (Section 5.1). Every resulting animation becomes a retargeting from our representation onto a polyline (or polylines) of the target model (Section 5.2).

5.1 Representation

In AniMesh, motions are stored as rotations over time for each skeletal bone. Due to the piece-wise rigidity of a bone skeleton, orientations change only at skeletal nodes, and are therefore piece-wise constant along the source polyline. In order to facilitate the retargeting of motions onto target polylines of any kind, we propose to resample source motions at new node locations by distributing

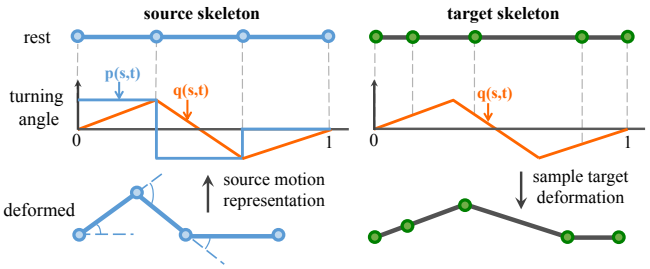


Figure 4: *2D analogy of motion representation and sampling. The source and target polylines' rest poses are shown in the top row. At time t, the source polyline is deformed (bottom row, left). Its animation (middle row, left) is represented as p(s,t) at time t, stored as the sum of turning angles for each bone, and q(s,t), our proposed interpolation of p(s,t). Using a unit arc-length parameterization, the target polyline is deformed (bottom row, right) by sampling from the source q(s,t) for each bone.*

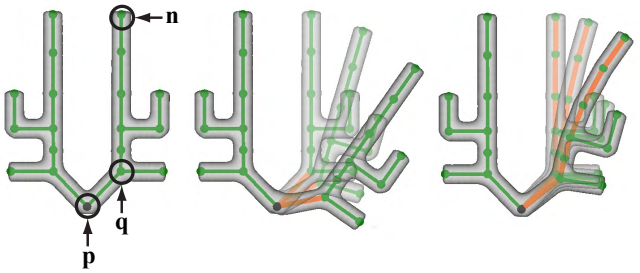


Figure 5: *The animator can choose how far motion is retargeted along a skeletal chain. Left: the target shape at rest pose. Middle: the animator can terminate the skeletal chain at the sub-tree root q; the entire sub-tree rotates rigidly. Right: alternatively, he can extend the skeletal chain along the longest branch of the sub-tree to a leaf node n.*

rotations along the bones in a piece-wise linear manner. Although it is possible to use higher-order interpolation, we find this representation to be simple, efficient, and effective in all of our experiments. Figure 4 (middle row) shows an example of this representation in 2D: turning angles $p(s,t)$ are stored piece-wise constant along the bones. By representing them using a piece-wise linear function $q(s,t)$, we can compute the turning angle anywhere along the polyline. To bring this analogy into 3D, and in order to map the animation to target polylines in a view-dependent fashion, we must fix the coordinate system. For each animated source polyline, its local (right-handed) basis is defined as follows: the z-axis points in the direction of the first bone; the y-axis is perpendicular to both the z-axis and the viewing direction, and points in the direction $< 90°$ from the capture device's up vector.

5.2 Retargeting

Using this representation we can retarget motions by sampling from $q(s,t)$ along the target polyline. Figure 4 illustrates this process using a 2D analogy: given a unit arc-length parameterization of source and target polylines, for each target bone, we use its parameter values at the two end points to sample the linear rotation function $q(s,t)$ from the source motion and set its rotation (relative to its parent) to be the difference between the sampled turning angles. In 3D we use the difference between sampled quaternions, and compute samples of $q(s,t)$ as needed by quaternion slerp [Shoemake 1985]. The target polyline after retargeting is shown in the bottom right of Figure 4. Note how the total turning angle of the source is preserved and the target polyline displays a similar shape to the source after retargeting in the bottom row of Figure 4.

With this retargeting method, we now need to decide on the target polyline onto which we will transfer the source motion. As mentioned in Section 3, the animator can manually specify the target polyline. However, in case of retargeting onto the top match from Section 4.2, the target polyline will not, in general, be uniquely defined. In Figure 5, our algorithm picked the sub-tree rooted at p in direction of q as one of the targets. Before transferring the source motion, we unfold the sub-tree, and then decide on one of two strategies for finding the target polyline: the animator can either have the polyline stop at q (Figure 5, middle) and rotate the entire sub-tree rigidly, or can choose to extend the matched polyline along the longest chain of the sub-tree, which terminates in a leaf n (Figure 5, right); the other branches rotate rigidly.

When transferring the motions, we want to maintain a consistent up direction between the source and the target polyline. Thus, similar to the source polyline, we define the right-handed basis of the target polyline view-dependently as the z-axis pointing in the direction of the first bone, and the y-axis being perpendicular to the z-axis

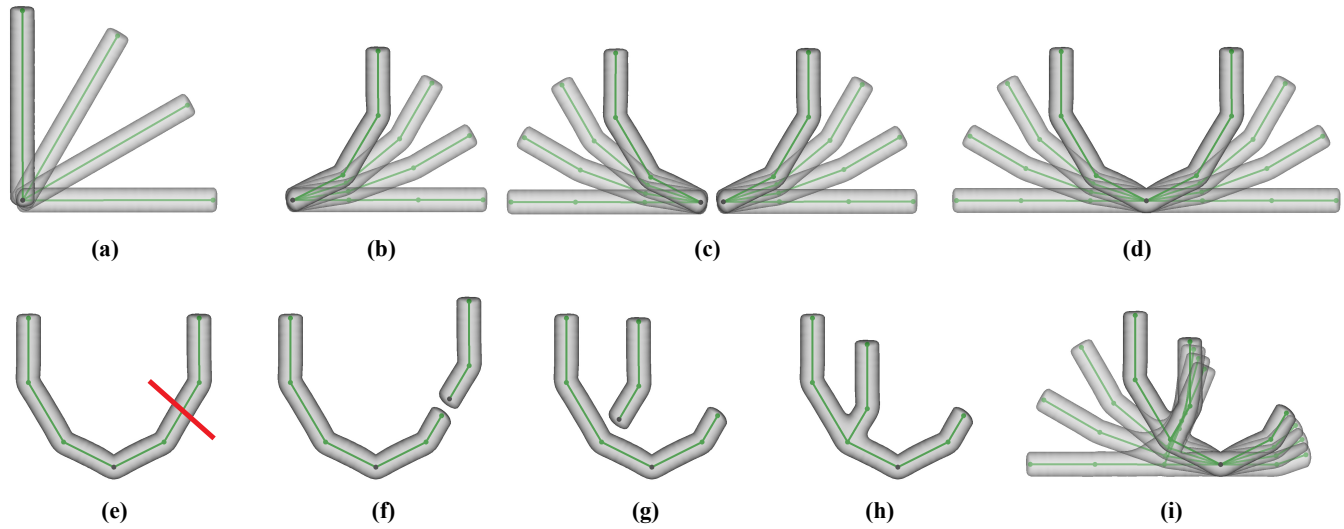


Figure 6: *An example shape editing sequence with animations. Poses at time $t = 1s$ are shown solid; poses at $t = [0, 0.33, 0.66]$ are shown semi-transparent (these intermediate poses are not shown in (e)-(h) for clarity of the modeling operations). (a) Shape with a single bone and a 90° counter-clockwise rotation from the rest pose. (b) The animator adds two new nodes, and the modified skeleton snaps to a new animation sequence; the total rotation along the polyline is preserved. (c) The animator makes a copy of the current shape and rotates it by 180°. (d) After merging the two shapes, the original motions are preserved. (e) Cutting at time t results in two shapes (f), while retaining bone orientations. (g) The cut-off part is merged to a different grafting node; the resulting shape (h) is consistent with the poses before the merge, and the original motions are preserved (i). (The complete modeling session is shown in the accompanying video.)*

and the viewing direction, with a positive inner product with the up vector of the current camera. The user can also manually flip the y-axis to change the transferred rotations.

6 Motion preserving shape editing

With skeleton matching and motion representation in place, we introduce the key element which enables our non-destructive workflow – motion preserving shape editing by incorporating motion retargeting into the editing process.

Most existing motion retargeting methods are a one-time process: once the animator decides on a transfer, the motion data is applied to the target skeleton; any modeling operations to the target model (e.g. deformation, cut, merge) at a later stage require that the animation be re-mapped onto the modified shape. Using our motion representation, this re-mapping is no longer necessary: for a skeletal polyline with animations attached, we can easily preserve its motions during shape editing by maintaining and updating its mapping into the source animation. Because editing operations may involve changes to the model as well as its animations, we must distinguish between *modeling transformations* (rotations introduced through skeletal deformation/posing operations) and *animation transformations* (rotations from retargeting source motion data).

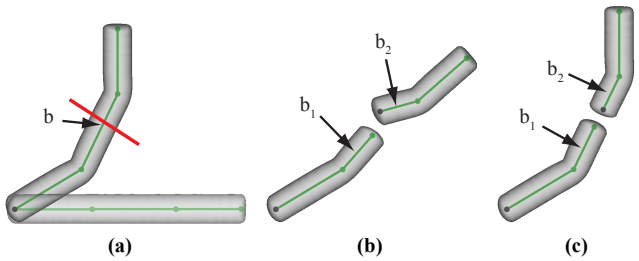


Figure 7: *Cutting operation at time t. (a) The shape has an animation attached to it; its rest pose is shown semi-transparent. The animator cuts bone b at time $t \neq t_0$. (b) Without adjustment of the modeling transformations, retargeted motions introduce a rotation discontinuity in the bone orientations of b_1 and b_2. (c) Adjusting the modeling transformation of the bones avoids this discontinuity.*

Adding and removing nodes. If the animator inserts a new node in the target polyline, we sample $q(s, t)$ at the new parameter value and update the adjacent bone rotations using aforementioned quaternion differences; node removal proceeds in similar fashion. Resampling the interpolated source motion $q(s, t)$ and computing bone rotations in this way guarantees that the total rotation along the polyline is invariant. Thus, removal of these nodes at a later date restores the original motion. The animator can insert new nodes at any time (Figure 6b).

Cutting. After a cutting operation, a polyline is separated into two parts. Figure 7 illustrates this process: bone b is cut into b_1 and b_2. A new node is added at the cutting position for both resulting polylines, along with its arc-length parameter value $s \in [0, 1]$. For the separated part, b_2, the cut node is the new root.

Intuitively, the world-space orientations of both b_1 and b_2 should remain unchanged after the cut. However, this is not the case for an animated polyline. Due to the change in arc-length parameter values, both b_1 and b_2 only receive a fraction of the retargeted rotation from the original bone b. This results in a discontinuity in their orientations (Figure 7b). We alleviate this by carefully adjusting their modeling rotations: for each bone affected by the cut (b_1, b_2), we compute the quaternion difference between its animation transformation and that of the un-cut bone (b), and add in (i.e. right-multiply) this difference onto its modeling transformation. This non-trivial transformation redistrubution ensures consistent bone orientations (Figure 7c).

Merging. We illustrate the process in Figure 8: the animator can specify a grafting node p, grab one shape and merge it with another. The root node of the shape being grafted onto r is retained for the final merged shape. Similar to the cutting operation, we need to

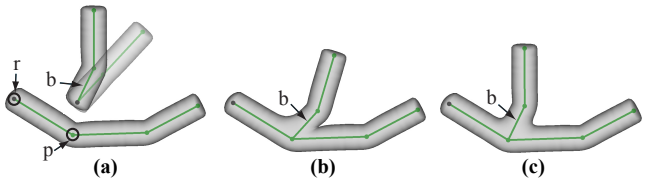


Figure 8: *Merging operation at time t. (a) The to-be-grafted shape with bone b has an animation attached to it; its rest pose is shown semi-transparent. (b) Without adjusting the modeling rotation of b, retargeted motion after the merge results in a different orientation for the shape being attached. (c) Adjusting the modeling rotation retains the original orientations.*

adjust the modeling transformation for the bone b directly being attached to p to avoid orientation discontinuities after the merge (Figure 8b).

To do this, we calculate the world-space orientation of the bone directly preceding p, accumulating both modeling and animation transformations starting from its root r, and remove (i.e. invert and left-multiply) this rotation from the modeling transformation of b. This retains the orientation of b before the merge (Figure 8c).

7 Results

Using AniMesh, animators can quickly create expressive animation sequences by iterating on both the model and its motion. A complete animation session, including transferring and layering live motion data from Kinect and LeapMotion, can be found in the accompanying video.

Since the representation of animation in AniMesh does not depend on the source of motion, live motion captured data can be replaced by canned (pre-recorded) animations. In Figure 9, the animator manually selects a skeletal polyline on the squid tentacle, transfers the canned source animation, and then makes copies of the shape as well as the attached animations. By merging the tentacles she can quickly generate composite animations.

With our non-destructive workflow, the basic operations of skeletal cut/merge and animation copy/paste provide powerful tools for creating complex animated shapes. Figure 10 shows the modeling and animation of a mantis. The user creates the torso and arms for a mantis from scratch. She maps some simple motions from an animation library onto the front arm, makes a copy of the arm along with its animation, and attaches them to the shape; she performs a similar operation for the front legs, then cuts off portion to reuse as the back legs. By integrating modeling with animation, the user need not worry about re-animating due to structural changes.

In order to gauge the accessibility of our system to novices, we showed AniMesh to a small group of children between 9 and 11 years old. We observed that they typically found the manipulation of timing with a timeline interface to be overly complex. Instead, it was much more straightforward for them to use Kinect or LeapMotion for direct mapping and animation. Figure 11 shows two children animating using the Kinect. They found that matching using their own poses was very intuitive, and were excited when their models started animating.

To evaluate how the simple operations of cut/merge, animation copy/paste and time shifting in AniMesh can make modeling and animation accessible to novices, we also invited 11 first-time

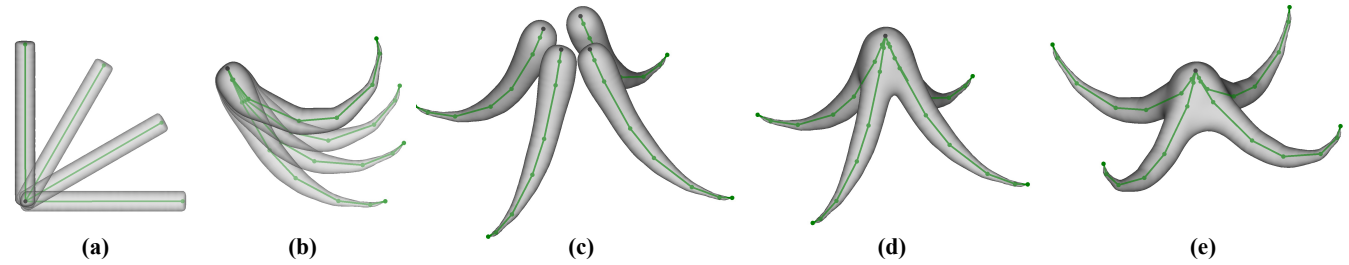


Figure 9: *Creating an animated squid with AniMesh. An existing motion (a) retargeted onto a single squid tentacle (b). The semi-transparent parts indicate the motion sequence. To re-use the existing animation, the user makes multiple copies of the animated tentacle, rotates them, and merges them (c). These merges result in a watertight squid raising its tentacles (e) from the rest pose (d).*

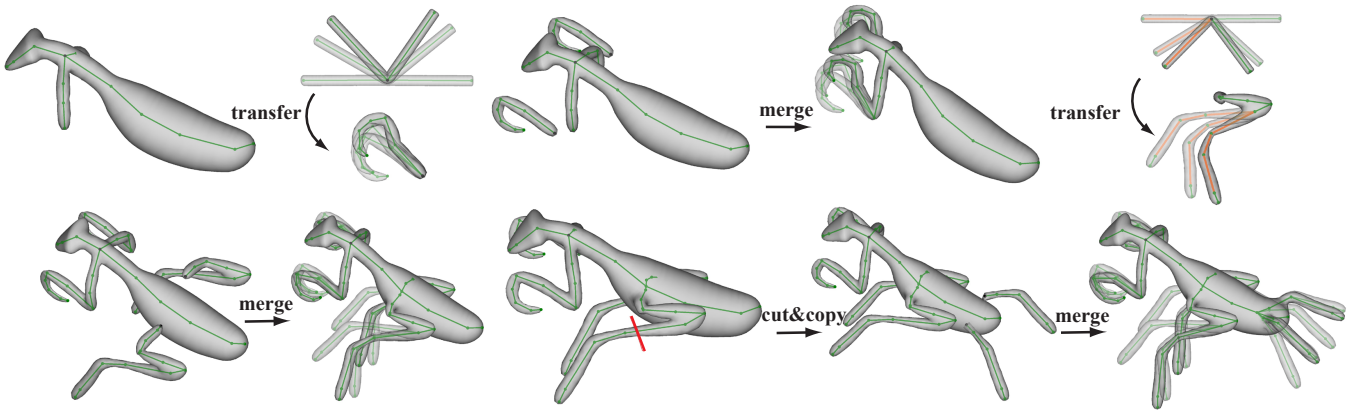


Figure 10: *Creating an animated mantis in AniMesh using only cut/merge and animation copy/paste. The semi-transparent parts indicate the motion sequence. A complete sequence of modeling, animation and iteration can be found in the supplementary video.*

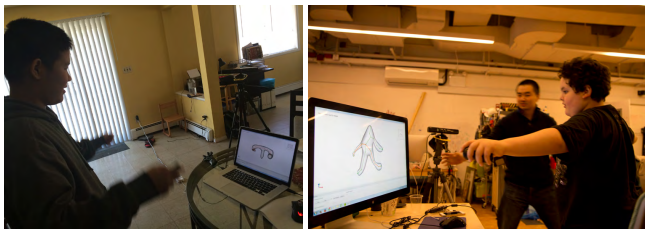


Figure 11: *Children modeling and animating with AniMesh.*

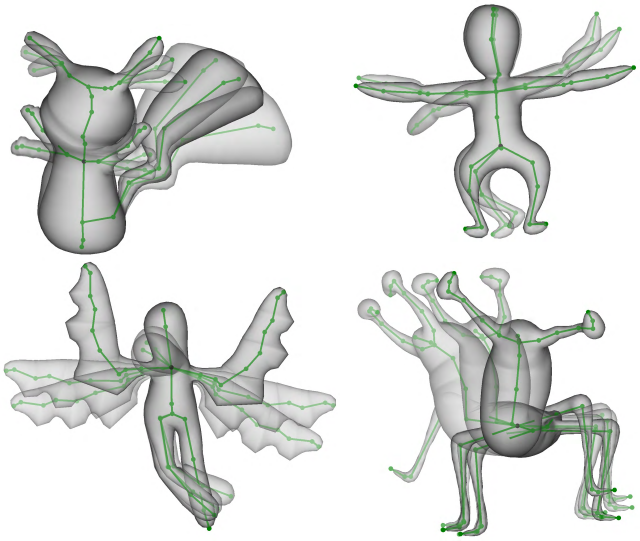


Figure 12: *Shapes modeled and animated using AniMesh. The semi-transparent parts indicate motion. These shapes took an average of 20 minutes. Please refer to the accompanying video.*

users, aged between 18 and 38, to participate in an informal user study. We trained them for approximately 20 minutes, and then allowed them to model and animate a shape without time limits. The subjects modeled their shapes from scratch, and were given access to a set of basic pre-recorded animations. None of the subjects had significant 3D modeling or animation experience. Two subjects were professional 2D artists, and another two subjects were indie game developers/designers. A selection of their modeling/animation results are shown in Figure 12.

When asked about the quality of their experience, most subjects reported that AniMesh was easy to use, and that the interleaved modeling and animation process helped them quickly explore and refine their designs. Some subjects attempted to create realistic animations, such as walking or running motions, and found this difficult to achieve. After our explanation of the design goal of AniMesh as a prototyping tool, they changed their design focus and had success with more cartoony character motions. One of the 2D artists appreciated "the flexibility in building the mesh, skeleton and the animation simultaneously," and enjoyed "the freedom in the transfer of motions from any existing shape parts." She also saw the potential of using more sophisticated human mo-cap data within our non-linear workflow. The other artist was impressed with how he could create an animated character in only 15 minutes (Figure 12 top-right). To quote the artist: "This is very useful for rapid prototyping. I can totally see it being used at a game jam to quickly create animation contents, such as crowd animations". One indie game designer concurred and emphasized how important rapid prototyping with AniMesh could be for the indie community, which usually has only limited time and resources, and where it is economically impractical to rent a motion capture system during development.

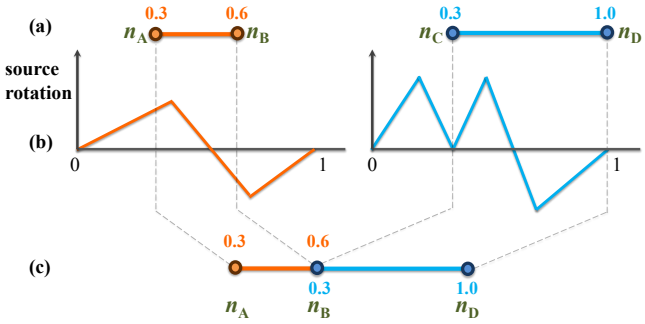


Figure 13: *(a) Merging two bones at n_B and n_C, each of which maps into a different source animation (b, orange and cyan). After the merge (c), the grafting node n_B has parameter 0.6 for one source motion and parameter 0.3 for another. Removing node n_B will invalidate the existing animation for the resulting, single bone.*

8 Discussion and future work

When comparing the geometry of source and target skeletons we measure their similarity based on their projections in screen space rather than world space. Allowing users to match against targets in world space is troublesome due to human difficulty understanding the orientation of 3D objects on a 2D display. Matching in screen space, on the other hand, is perceptually unambiguous. The screen space similarity metric, however, becomes degenerate when a bone is perpendicular to the screen. Fortunately, this case is uncommon in practice, and easily solved by rotating the model to a better perspective. In light of this, it would be interesting to investigate perceptual viewpoint preference [Secord et al. 2011] to automatically suggest good camera views for the matching (e.g. by maximizing the sum of projected bone lengths).

Despite the generality of our matching technique, users still find it easiest to match their skeleton to humanoid subsections of a given target and animate a complicated rig in multiple passes. It's likely this is an inherent psychological tendency, however it would be valuable to explore if our matching algorithm could be improved to encourage artists to animate characters in fewer sequential passes.

The choice to model animation as cumulative bone rotations was made based on our intuition that it naturally represented the intention of the user. Later, we confirmed through experiments that users were in fact very comfortable with this representation. On the other hand, there is still room to explore alternative methods of retargeting motion, such as transferring end effector positions between source and target, then using inverse kinematics to generate intermediate skeletal motion. This approach in particular solves foot-skate, and allows us to then explore other features such as root node translation. However, as currently designed, the animations created with AniMesh can be fine-tuned at a later stage by enforcing constraints for interactions with the environment [Gleicher 1998; Kovar et al. 2002].

After merging two skeletons, an ambiguity may arise: the grafting node maintains parameter values associated with different source animations (Figure 13). In our current implementation, deleting the grafting node n_B will leave both chains' motions undefined; we simply invalidate the motions attached to both chains in these cases. A suitable reparameterization may be able to address the issue.

In the future, we would also like to investigate using symmetry to apply animations to similar skeletal parts at the same time. For example, animating the multiple tentacles of a squid simultaneously. If rotational symmetry is detected, the animator would only need to specify the mapping to one tentacle, and the same motion could be automatically applied to the rest.

Acknowledgement

We would like to thank the anonymous reviewers for their valuable comments and suggestions. We also want to thank Bert Buccholz for his helpful discussions on the project. We greatly appreciate the efforts of our participants for the models and animations they created during the informal user study. This research is supported in part by the NSF (IIS-13-31001, IIS-1451198, and IIS-1453018) and a Google research award.

References

BARAN, I., AND POPOVIĆ, J. 2007. Automatic rigging and animation of 3D characters. *ACM Trans. Graph. 26*, 3 (July).

BOFF, K. R., KAUFMAN, L., AND THOMAS, J. P. 1986. *Handbook of Perception and Human Performance*. Wiley.

BOROSÁN, P., JIN, M., DECARLO, D., GINGOLD, Y., AND NEALEN, A. 2012. RigMesh: Automatic rigging for part-based shape modeling and deformation. *ACM Trans. Graph. 31*, 6 (Nov.), 198:1–198:9.

CHEN, J., IZADI, S., AND FITZGIBBON, A. 2012. KinÊtre: Animating the world with the human body. In *Proceedings of ACM UIST*, 435–444.

DONTCHEVA, M., YNGVE, G., AND POPOVIĆ, Z. 2003. Layered acting for character animation. *ACM Trans. Graph. 22*, 3 (July), 409–416.

GLEICHER, M. 1998. Retargetting motion to new characters. In *Proceedings of ACM SIGGRAPH*, 33–42.

HECKER, C., RAABE, B., ENSLOW, R. W., DEWEESE, J., MAYNARD, J., AND VAN PROOIJEN, K. 2008. Real-time motion retargeting to highly varied user-created morphologies. *ACM Trans. Graph. 27*, 3 (Aug.), 27:1–27:11.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3D freeform design. In *Proceedings of ACM SIGGRAPH*, 409–416.

JACOBSON, A., PANOZZO, D., GLAUSER, O., PRADALIER, C., HILLIGES, O., AND SORKINE-HORNUNG, O. 2014. Tangible and modular input device for character articulation. *ACM Trans. Graph. 33*, 4 (July), 82:1–82:12.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. *ACM Trans. Graph. 21*, 3 (July), 473–482.

KULPA, R., MULTON, F., AND ARNALDI, B. 2005. Morphology-independent representation of motions for interactive human-like animation. *Computer Graphics Forum, Eurographics 2005 special issue 24*, 343–352.

LEAPMOTION, 2015. Leapmotion. https://www.leapmotion.com/.

LEE, J., AND SHIN, S. Y. 1999. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of ACM SIGGRAPH*, 39–48.

MICROSOFT, 2015. Kinect. http://www.microsoft.com/en-us/kinectforwindows/.

MILLER, C., ARIKAN, O., AND FUSSELL, D. 2010. Frankenrigs: Building character rigs from multiple sources. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '10, 31–38.

MINE, M. R., BROOKS, JR., F. P., AND SEQUIN, C. H. 1997. Moving objects in space: Exploiting proprioception in virtual-environment interaction. In *Proceedings of ACM SIGGRAPH*, 19–26.

MONZANI, J.-S., BAERLOCHER, P., BOULIC, R., AND THALMANN, D. 2000. Using an intermediate skeleton and inverse kinematics for motion retargeting. *Comput. Graph. Forum 19*, 3, 11–19.

NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2007. FiberMesh: Designing freeform surfaces with 3D curves. *ACM Trans. Graph. 26*, 3 (July).

POIRIER, M., AND PAQUETTE, E. 2009. Rig retargeting for 3d animation. In *Proceedings of Graphics Interface 2009*, Canadian Information Processing Society, GI '09, 103–110.

POPOVIĆ, Z., AND WITKIN, A. 1999. Physically based motion transformation. In *Proceedings of ACM SIGGRAPH*, 11–20.

RHODIN, H., TOMPKIN, J., KIM, K. I., KIRAN, V., SEIDEL, H.-P., AND THEOBALT, C. 2014. Interactive motion mapping for real-time character control. *Computer Graphics Forum (Proceedings Eurographics) 33*, 2.

SECORD, A., LU, J., FINKELSTEIN, A., SINGH, M., AND NEALEN, A. 2011. Perceptual models of viewpoint preference. *ACM Trans. Graph. 30*, 5 (Oct.), 109:1–109:12.

SEOL, Y., O'SULLIVAN, C., AND LEE, J. 2013. Creature features: Online motion puppetry for non-human characters. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, SCA '13, 213–221.

SHIN, H. J., LEE, J., SHIN, S. Y., AND GLEICHER, M. 2001. Computer puppetry: An importance-based approach. *ACM Trans. Graph. 20*, 2 (Apr.), 67–94.

SHOEMAKE, K. 1985. Animating rotation with quaternion curves. In *Proceedings of ACM SIGGRAPH*, 245–254.

SUMNER, R. W., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Trans. Graph. 23*, 3 (Aug.), 399–405.

SUMNER, R. W., SCHMID, J., AND PAULY, M. 2007. Embedded deformation for shape manipulation. *ACM Trans. Graph. 26*, 3 (July).

VAN KAICK, O., ZHANG, H., HAMARNEH, G., AND COHEN-OR, D. 2011. A survey on shape correspondence. In *Computer Graphics Forum*, vol. 30, Wiley Online Library, 1681–1707.

VÖGELE, A., HERMANN, M., KRÜGER, B., AND KLEIN, R. 2012. Interactive steering of mesh animations. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '12, 53–58.

WATT, A., AND WATT, M. 1991. *Advanced animation and rendering techniques*.

YAMANE, K., ARIKI, Y., AND HODGINS, J. 2010. Animating non-humanoid characters with human motion data. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, 169–178.

YUMER, M. E., AND KARA, L. B. 2012. Co-abstraction of shape collections. *ACM Trans. Graph. 31*, 6 (Nov.), 166:1–166:11.