COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds (2013)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cav.1560

RESEARCH ARTICLE

Fast, automatic character animation pipelines

Andrew Feng[1], Yazhou Huang[2], Yuyu Xu[1] and Ari Shapiro[1]*

[1] USC Institute for Creative Technologies, Playa Vista, CA, USA
[2] University of California, Merced, Merced, CA, USA

ABSTRACT

Humanoid three-dimensional (3D) models can be easily acquired through various sources, including through online marketplaces. The use of such models within a game or simulation environment requires human input and intervention in order to associate such a model with a relevant set of motions and control mechanisms. In this paper, we demonstrate a pipeline where humanoid 3D models can be incorporated within seconds into an animation system and infused with a wide range of capabilities, such as locomotion, object manipulation, gazing, speech synthesis and lip syncing. We offer a set of heuristics that can associate arbitrary joint names with canonical ones and describe a fast retargeting algorithm that enables us to instill a set of behaviors onto an arbitrary humanoid skeleton on-the-fly. We believe that such a system will vastly increase the use of 3D interactive characters due to the ease that new models can be animated. Copyright © 2013 John Wiley & Sons, Ltd.

KEYWORDS
animation; graphics; system; retargeting

***Correspondence**
Ari Shapiro, USC Institute for Creative Technologies, Playa Vista, CA, USA.
E-mail: shapiro@ict.usc.edu

1. MOTIVATION

Three-dimensional (3D) characters are commonly seen in video games, feature films, mobile phone applications and websites. The generation of an expressive 3D characters requires a series of stages, including the generation of a character model, specifying a skeleton for that model, deforming the model according to the movement of the skeleton, applying motion and control algorithms under a framework, and finally instructing the character to perform. Each of these processes requires a different skill set. For example, 3D models are generated by digital modelers or through hardware-based acquisition, whereas animators create or apply motion to the characters.

Thus, although many high quality assets such as humanoid models or motion capture data can be readily and inexpensively acquired, the integration of such assets into a working 3D character is not automated and requires expert intervention. For example, after motion capture data is acquired, it then needs to be retargeted onto a specific skeleton. An acquired 3D humanoid model needs a skeleton that satisfies the constraints of a real-time game system, and so forth. Modern game engines provide a means to visualize and animate a 3D character, but require assembly by a programmer or game designer. The complexity of animating 3D virtual characters presents an obstacle for the end user, who cannot easily control a 3D character without the assistance of specialists, despite the broad availability of the models, assets and simulation environments.

To address this problem, we present a system that allows the rapid incorporation of high-fidelity humanoid 3D models into a simulation. Characters introduced to our system are capable of executing a wide range of common human-like behaviors. Unlike a traditional pipeline, our system requires no intervention from artists or programmers to incorporate such characters after the assets have been generated. Our pipeline relies upon two key automated processes:

(1) An automated skeleton matching process; skeletons are examined to find a match between the new skeleton and one recognized by the simulation. Such a process looks for similarly named joints, as well as relies on expected topology of humanoid in order to recognize similarly functioning body parts.
(2) A retargeting process that can transfer high quality motion sets onto a new character without user intervention.

Figure 1 summarizes the pipeline of our system for automatic animation transfer [1]. The pipeline consists of two main stages. The skeleton analysis stage takes two skeletons as input and matches them by remapping their joint names and aligning their joint local frames. The motion retarget stage then converts the input motion for the source

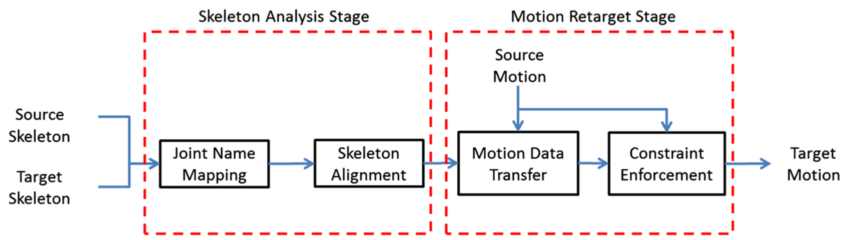


Figure 1. The overview of our animation transfer process. The source skeleton and target skeleton are first analyzed to remap their joint names and realign their joint local frames. Then at the retarget stage, a motion from source skeleton could be converted to fit the target skeleton via motion data transfer and constraint enforcement.

skeleton so it can be used directly on the target skeleton. Note that the retarget stage can either be performed offline to generate new motions that are suitable for target skeleton, or directly apply the source motion on the target skeleton by converting joint angles on the fly.

In addition, the virtual character's capabilities are generally based on two different sources:

(a) A set of controllers that can generate motion by means of rules, learned models, or procedurally-based methods, and
(b) A set of behaviors generated from animation data that can be parameterized across various dimensions, such as running speed for locomotion, or reaching location for interaction with other 3D objects.

2. RELATED WORK

The first stage of our system utilizes an automated mapping process, which uses a set of heuristics for mapping an arbitrarily named humanoid skeleton onto a common skeleton with familiar names. To our knowledge, no such algorithm has been previously published. Many other methods for registering skeletons require matching names or manual annotations [2]. At the time of this writing, [3] demonstrates a process by which a skeleton can be automatically registered, but no technical details are provided regarding underlying algorithms and robustness. In addition, we are aware of systems such as the work of Arikan and Ikemoto [4], which attempt to automate the acquisition of motion and models, but have not seen any details regarding the skeleton rig recognition step.

The second stage of our system utilizes a fast retargeting system to generate animations appropriate for a particular skeleton. Retargeting has been an area of much research in the animation community since Gleicher [5]'s work, which uses optimization and low-pass filtering to retarget motion. Many other retargeting algorithms use various approaches: Kulpa [6] retargets motion by using angular trajectories and then, solve several body areas, Lee [7] uses a hierarchical approach to retargeting, and Mozani [8] uses an intermediate skeleton and inverse kinematics (IK) to handle retargeting between skeletons with different topologies. Kulpa [6] retargets motion through a morphology-independent representation by using angular trajectories and then, solving several body areas. Taku [9] uses spatial relationships for motion adaptation, which can handle many contact-based motion retargeting problems. Zordan [10] retargets optical data directly onto a skeleton via a dynamics-based method. Shin [11] uses an online retargeting method via an analytical IK method that prefers the preservation of end effector values. Choi [12] uses a Jacobian-based IK method for online retargeting.

Our retargeting system attempts to find foot plants in order to better retarget movement. An online foot plant detection and enforcement method is presented in Glardon's work [13]. By contrast our retargeting method does not detect foot plants online, and does not modify the length of limbs as in [14] so as to be compatible with many game and simulation skeleton formats. Similar to our goals, the work in [15] is focused on retargeting to creatures with a varying morphology, such as differing number of legs, tails, or the absence of arms. The system described in that work relies heavily on IK in performing online retargeting based on semantic descriptions of movement. By contrast, we are interested a relatively fast online retargeting process to transfer high-quality motions onto humanoid characters that cannot be achieved via simple walk cycles and reaching constraints. Miller *et al.* [2] develops a system to automatically assemble a best-fitting rig for a skeleton. By contrast, our system assumes the skeleton and model have already been connected and focus on the use of such skeleton in real-time simulations.

The characters in our system can be instructed to perform certain behaviors using the Behavioral Markup Language (BML) [16]. BML provides a high-level XML-based description of a set of tasks that can be synchronized with each other. Many systems have been developed to utilize BML, such as EMBR [17], ELCKERLYC [18], BEAT [19] and GRETA [20] in addition to our system, SMARTBODY [21,22]. However, to our knowledge, no other BML-based systems besides our own have implemented extensive character locomotion capabilities or generic capabilities such as object manipulation [23], which are associated with large sets of behaviors. Because the BML specification emphasizes speech, head movements, and gestures, most BML-compatible systems emphasize only those features.

3. AUTOMATIC SKELETON JOINT MAPPING

One of the challenges of using an off-the-shelf character model is that the user has to first set up a joint mapping table to comply with the skeletal system and motion data used for the target system/application. This step is critical for many motion parameterization procedures such as retargeting, and although being a trivial task, it is commonly performed by hand.

The joint mapping is a necessary step for online motion retargeting, because it establishes the proper relationship between source and target skeletons. Our assumption is that the input skeletons should have similar anatomy to a humanoid character. Thus, they should have essential joints such as spines, elbows, and knees that are required to perform various actions. Although it is possible for the two skeleton to have very distinct configurations and number of joints, as far as these essential bones present, the joint mapping would allow them to be utilized during retargeting.

The key advantage of performing the skeleton mapping during preprocessing is that the run-time retargeting step can be simplified to allow real-time performance for behavior transfer. Without proper joint correspondences, the retargeting step would require full analysis of input motions and skeletons to compute the optimal joint angles for the target skeleton [5]. Although it can yield better retargeting results, it requires nonlinear optimization and thus, is not suitable for real-time applications. Note that existing commercial tools such as (Autodesk, San Rafael, California) MotionBuilder [24] also requires such joint mapping step to perform retargeting.

In this submission, we propose a heuristic-based automatic skeleton joint mapping. Our method utilizes the skeleton hierarchy structure and symmetries, combined with keyword searching to help determine certain key joints in the hierarchy. We have successfully validated our automatic mapping method using character skeletons from various popular sources (mixamo.com, rocketbox-libraries.com, turbosquid.com, axyz-design.com, 3DSMax, and MotionBuilder), results shown in Figure 2.

Our goal is to map a list of arbitrary joints from any user-defined biped skeleton to the set of canonical joints on the target skeleton inside our character animation system. Figure 3 shows the final mapping result to be achieved from left-side mapped to the right side. The left side example follows MotionBuilder [24] standard skeleton joint naming convention, and the right side is the corresponding names in our SmartBody standard skeleton. We do not intend to map all the joints, and in many cases, not all joints can be mapped easily. We map only a basic set of joints that would enable most of our controllers to drive user-defined characters for behaviors such as gaze, reaching, and locomotion.

Algorithm 1 Search routine for base joint.

```
1.  while $i \le max\_search\_depth$ do
2.      $J \leftarrow$ skeleton.joint(i)
3.      switch ($J$.num_children())
4.      case 2:
5.          if $J$ has 2 symmetrical children then
6.              return MAP(base, J)
7.          end if
8.      case 3:
9.      if $J$ has 2 symmetrical children then
10.         return MAP(base, J); MAP(spine)
11.     end if
12.     end switch
13. end while
14. return BASE_NOT_FOUND
```

The mapping is largely based on heuristics and is specifically adapted to our system. The first step is to find the character's base joint. We only consider the situation where the input skeleton is biped, in which case the base is usually defined as the parent of spine and two legs. Figure 4 (top left) generalizes some of the variations found in our testing skeletons, and the routine is outlined in Algorithm 1. Once the base joint is found, our algorithm tries to map the remaining key joints based on the symmetry/hierarchy of the canonical target skeleton and the assumption that

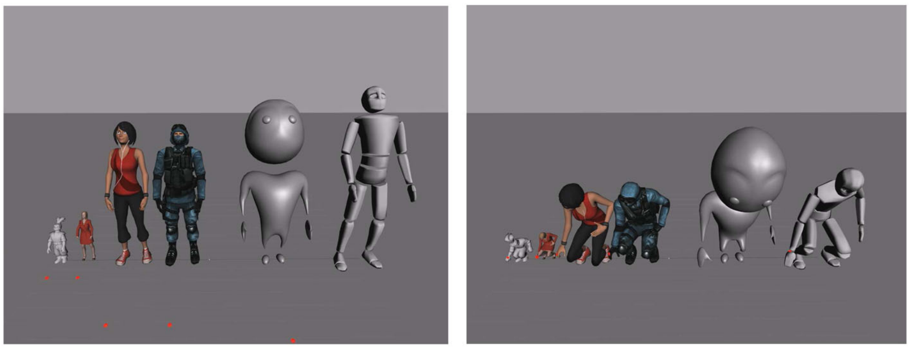
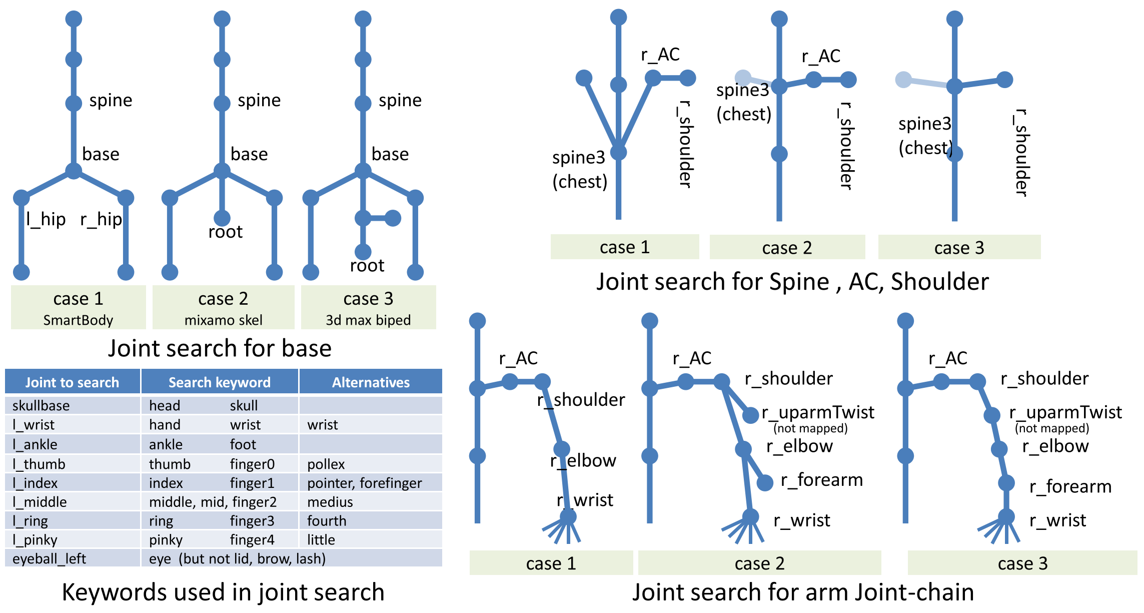


Figure 2. A set of characters from many different sources are automatically retargeted and registered into our system. The characters can now perform a number of tests with controllers and parameterized motions in order to insure that the behavior has been properly transferred: gazing, object manipulation, locomotion, head nodding, and so on.

```
Reference
 Hips
   Spine
     Spine1
       Spine2
         Spine3*
           Neck
             Head
               LeftEye
         LeftShoulder
           LeftArm
             LeftArmTwist*
               LeftForeArm
                 LeftForeArmTwist*
                   LeftHand
                     LeftHandThumb1∼4
                     LeftHandIndex1∼4
                     LeftHandMiddle1∼4
                     LeftHandRing1∼4
                     LeftHandPinky1∼4
   LeftUpLeg
     LeftLeg
       LeftFoot
         LeftToeBase
           LeftToeBase2
             LeftToeBase3*
   ...[right side joints] ...
```

```
base
 spine1
   spine2
     spine3
       spine4
         spine5
           skullbase
             eyeball_left
     l_acromioclavicular
       l_shoulder
         l_elbow
           l_forearm
             l_wrist
               l_thumb1∼4
               l_index1∼4
               l_middle1∼4
               l_ring1∼4
               l_pinky1∼4
 l_hip
   l_knee
     l_ankle
       l_forefoot
         l_toe
 ...[right side joints] ...
```

Figure 3. Final mapping result achieved by our system—left side is given as an example following MOTIONBUILDER naming convention, and right side is the corresponding joint names in our system. * denotes joint (if exists) is skipped as it is not handled by our system.



Joint to search	Search keyword		Alternatives
skullbase	head	skull	
l_wrist	hand	wrist	wrist
l_ankle	ankle	foot	
l_thumb	thumb	finger0	pollex
l_index	index	finger1	pointer, forefinger
l_middle	middle, mid, finger2		medius
l_ring	ring	finger3	fourth
l_pinky	pinky	finger4	little
eyeball_left	eye (but not lid, brow, lash)		

Figure 4. An illustration of various configurations generalized from testing skeletons for certain key joints and joint-chain mapping using heuristics.

source skeleton will have similar properties. Here, we could only show a small portion of this procedure, Figure 4 and Algorithms 2 and 3 outline part of the search routines for spine/chest and arm joint-chain, respectively; however, more complicated cases are also handled. As an example, if two joints are found sharing the same parent joint (Spine #), both have the same depth also the same number of children joints, the algorithm will assume they are either acromioclavicular or shoulder, and then, determine left/right using their joint names. Another example is that based on the depth of shoulder and wrist in the hierarchy, the heuristic determines if twist joints are present in-between and estimates the mapping accordingly. In certain cases, the heuristics may rely on keyword search inside joint names to determine the best mapping, but switches to purely hierarchy-based mapping when not successful. Please refer to our code base (Section 7) for details of the mapping procedure. Because the mapping is based on the heuristics, it may not work in all cases. Therefore, characters with uncommon hierarchy or asymmetrical hierarchy may break the algorithm. For example, when a character has extra joint branches such as wings or tails, it would be difficult for our method to correctly identify the arms and legs. Another limitation of our heuristics is that it relies on reasonable joint naming conventions in the skeleton to identify left and right joints. For example, if the joints in a skeleton are named in arbitrarily as 'joint1', 'joint2', and so forth, our algorithm would not be able to guess the left or right arms from joint names. For such difficult cases, the user needs to manually complete the mapping starting with the partial mapping table generated by the algorithm.

Algorithm 2 Search routine for spine, chest, acromioclavicular, and head joints.

```
 1. $J \leftarrow$ base
 2. while $J \leftarrow J.child()$ do
 3.     if $J.num\_children() \geq 2$ then
 4.         MAP(Spine4, J) {chest joint}
 5.         break
 6.     else
 7.         MAP(spine#, J)
 8.     end if
 9. end while
10. if $J$ has 2 symmetrical children then
11.     MAP(AC, J.child())
12. end if
13. if $J.child().name() = "Head"$ then
14.     MAP(skellbase, J.child())
15. end if
```

4. RETARGETING PROCEDURES

The motion retargeting process works by transferring an example motion set from our canonical skeleton to a custom skeleton provided by the user. The retargeting process can be separated into two stages. The first stage is to convert the joint angles encoded in a motion from our canonical skeleton to the custom skeleton. The second stage is to enforce various positional constraints such as foot positions to remove motion artifacts such as foot sliding.

4.1. Motion Data Transfer

The goal of this stage is to transfer the motion data such as joint angles from a source skeleton to a target skeleton. Joint values can be directly copied over for skeletons with aligned local frames and initial T-poses. However in most cases, a skeleton provided by the user tends to have different setup and default pose from our canonical skeleton. Therefore, we first need to align the default pose between the target skeleton S_d and the source skeleton S_r. This is performed by recursively rotating each bone segment in target skeleton to match the global direction of that segment in source skeleton at default pose (Figure 5 left) so that the target skeleton is adjusted to have the same default pose as the source skeleton.

Once the default pose is matched, we address the discrepancy between their local frames by adding suitable prerotation and post-rotation at each joint in target skeleton. Specifically, given a joint b_i, with its global rotation R_d^G and initial local rotation q_d^{init} when in default T-pose, we reorient its local frame as

$$q_d = q_d^{init} \ R_d^{G^{-1}} \ q_g \ R_d^G \quad (1)$$

where q_t is the actual local rotation after reorientation, and q_g is the standard rotation that complies with the default global frame. In other words, the original local frame of b_i is reoriented to align with the default canonical global frame as shown in Figure 5 right, for example, a left 30° turn around y-axis in Y-Up global frame simply means setting $q = quat((vec(0,1,0), 30)$ without considering the initial rotation of b_i. Because our canonical skeleton already has all of its joint local frames aligned with the global frame, this in turn aligns joints in both skeletons into the same local frames. Therefore, the motion data transfer can now be carried out trivially by copying the joint rotations to the target skeleton. Similarly, the root translation p_r can also be transferred to the target skeleton by scaling it according to the length of legs between two skeletons. The scale factor s_r is computed as $s_r = \frac{l_t}{l_s}$, where l_t is the leg length of target skeleton, and l_s is that of source skeleton. For motions created specifically for skeletons with noncanonical alignments, we reverse the reorientation process as

$$q_g = R_r^G \ q_r^{init^{-1}} \ q_r \ R_r^{G^{-1}} \quad (2)$$

to make these motions become aligned with default global frame, which can be directly applied to any skeleton after realignment in a very straightforward fashion.

4.2. Constraint Enforcement

Once motion data is transferred, they would serve as a rough approximation to enable the target skeleton with

Algorithm 3 Search routine for arm joint-chain.

Arm_Joint-Chain_Search (sk)

```
1.  $J \leftarrow$ acromioclavicular (AC)
2.  while $J \leftarrow J.child()$ do
3.      if $J$ has 5 children then
4.          MAP(wrist, J)
5.      else if $J.num\_children() = 0$ then
6.          $J \leftarrow J.parent()$
7.          MAP(wrist, J)
8.      end if
9.  end while
10. if not wrist.mapped() then
11.     return WRIST_NOT_FOUND
12. end if
13. $J_1 \leftarrow shoulder$ ; $J_2 \leftarrow wrist$
14. switch ($J_2.depth - J_1.depth$)
15. case 2:
16.     uparm $\leftarrow$ shoulder
17.     MAP(uparm);MAP(elbow)
18. case 3:
19.     MAP(uparm);MAP(elbow)
20. case 4:
21.     MAP(uparm);MAP(elbow)
        if forearm then MAP(forearm)
22. case 5:
23.     MAP(uparm);MAP(elbow);
        MAP(forearm)
24. end switch
```

various behaviors such as locomotion. However, the transferred motion may not work perfectly on the target skeleton due to different limb lengths, which may result in foot sliding artifacts, and so on. This problem could be seen in many kinds of motions after naive data transfer, but is mostly visible among locomotion sets. In order to alleviate these artifacts, we apply IK to enforce the foot plant constraint in the transferred motions. The IK method we use is based on Jacobian pseudo-inverse, $\Delta\Theta = J^{+}\Delta\mathbf{x} + (I - J^{+}J)\Delta\mathbf{z}$, where $J^{+} = J^{T}(JJ^{T})^{-1}$ is the pseudo-inverse of Jacobian matrix J, $\Delta\mathbf{x}$ is the offset from current end effector coordinates to target coordinates $\mathbf{x_r}$, and $\Delta\mathbf{z}$ is the desired joint angle increments toward target pose $\mathbf{z} = \tilde{\Theta}$. The former IK method deforms an input pose to satisfy the end effector constraint, while maintaining the target pose $\mathbf{z}$ as much as possible. The key advantage of Jacobian method is that we can solve for all joint angles at once for the whole skeleton hierarchy instead of computing each joint chain individually. This is useful when two end effectors share some joints in their joint chains. For example, when the character is bending down to reach for a target with both hands, the spine joints need to be adjusted so both hands can be placed at target positions. It is more difficult to obtain such a solution if we solve each chain separately.

Jacobian method is known for its singularity when the target is out of the reach. In this situation the Jacobian becomes near singular, and the resulting joint angles become unstable. To alleviate this problem, we adapt a damping term $\lambda^2 I$ so $J^{+} = J^{T}(JJ^{T} + \lambda^2 I)^{-1}$. The damping term helps keep the matrix nonsingular even when the skeleton pose are in a singular configuration. The damping parameter is set to $\lambda = \alpha h$, where h is the height of character, and α is scaling factor. In our experiment, we found that setting $\alpha = 0.05$ provides a good balance for accuracy and stability. However, when the target is very far away, the Jacobian matrix could still be close to singular even with the damping term. Thus, we also force the target position to be within reach of the character by projecting it to a bounding sphere, that is, centered at the skeleton root and has the radius equals to the length of joint chain from end effector to root. This effectively avoids the target position to be too far away to remove the effect of damping term.

We apply this IK method at each motion frame in the locomotion sequences to ensure the foot joint is in the same position during the foot plant stage. If desired, the same method can be also applied on upper body to enforce hand placement constraints for gesturing or reaching. Previous methods exist for detecting and fixing foot

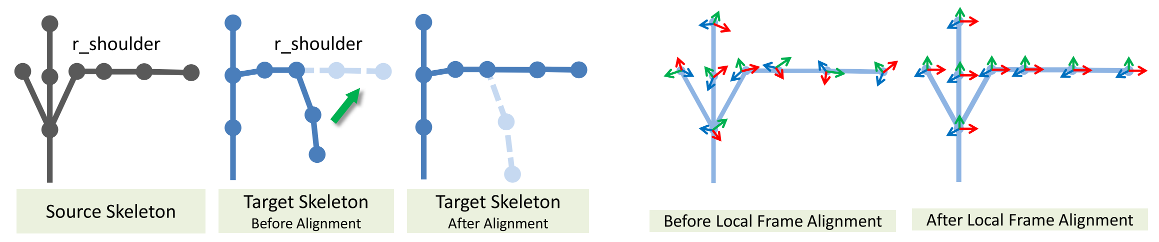


Figure 5. Left side shows alignment of a bone segment between two skeletons so that target skeleton matches the pose of source skeleton. Right side shows reorientation of joint local frames so that they align with the canonical world frame, which enables straightforward transfer of motion data from source to target skeleton.

sliding [13,14]. They mostly work by finding a time range over which the foot plant occurs and enforce the foot plan during that period. Additional smoothing is usually required to ensure that the constraint enforcement does not create popping artifacts in the motion. Through our experiments, we found that it is difficult to robustly detect foot plant range across different type of motions. Also, without careful consideration, smoothing may create more motion artifacts if foot plant is not correctly found. Because we assume that the original motion is smooth and does not contain foot sliding, we choose to warp the original motion trajectory and enforce constraints over the whole trajectory. Let $p_r^G(t)$, $p_d^G(t)$ be the global foot position trajectory for source and target skeleton. We create a new trajectory for target skeleton by warping the original trajectory using the following equation:

$$p'_d(0) = p_d^G(0)$$
$$p'_d(t+\delta t) = p'_d(t) + s_r\left(p_r^G(t+\delta t) - p_r^G(t)\right)$$

where p'_d is the new target trajectory, and s_r is the scaling factor based on leg length from the previous section. The former equation warps the foot trajectory from original skeleton based on the scale of target skeleton. The method was proven to work well during our experiments on various skeletons with different limb lengths and proportions.

5. ONLINE RETARGETING

The former procedures can be applied to process all the example motions to create a new set of motions for each new character. However, processing through all example motions may be time consuming when there are many unique characters. Online retargeting provides an efficient way to quickly apply original motions for a new character on-the-fly. Instead of preprocessing all example motion, it acts as a animation filter to recompute correct joint angles for the new character at run-time. This saves the preprocessing time in exchange for a run-time performance penalty. We are able to develop the online retargeting capability in our system by modifying the aforementioned retargeting procedures.

5.1. Online Motion Data Transfer

We separate the motion data transfer into the stage that only needs to be carried out once, and the stage that needs to be carried out during every motion frame. First, we precompute and store the joint mapping information for each skeleton. Then for each skeleton pair S_r and S_d, we compute their skeleton alignment, and store the alignment rotations that would transform the joint local rotation q_r in S_r to the suitable local rotation q_d in S_d. Specifically, because from Equations (1) and (2),

$$q_d = q_d^{init}\ R_d^{G^{-1}}\ R_r^G\ q_r^{init^{-1}}\ q_r\ R_r^{G^{-1}}\ R_d^G$$

we can define the alignment rotations as $q^{pre} = q_d^{init}\ R_d^{G^{-1}}\ R_r^G\ q_r^{init^{-1}}$ and $q^{post} = R_r^{G^{-1}}\ R_d^G$. Therefore, given a new q'_r for S_r, we can transfer the joint rotation to S_t via $q'_d = q^{pre}\ q'_r\ q^{post}$. Because this motion transfer is very efficient as it involves only two additional quaternion multiplication for each joint, we can readily apply it for online motion transfer with very little performance penalty. For a simple motion playback, we apply this formula directly to output a new motion for the new skeleton. For motion blending, naively transforming the joint rotation for each example motion on the fly would be expensive if there are many example motions. So we choose to combine the example motions normally in motion blending, and only transform the joint rotation before we output the blending results.

5.2. Online Constraint Enforcement

Constraint enforcements require global foot plant positions for solving IK. In order to provide this information, we preprocess each motion and store the local foot position trajectory $p_r^l(t) = p_r^G(t)\ M_r^{-1}(t)$ for the source skeleton, where $p_r^G(t)$ is the global foot position, and $M_r(t)$ is the global base transformation. The global foot trajectory of target skeleton S_d can be computed as $p_d^G(t) = s_r\ p_r^G(t)\ M_d(t)$ where s_r is the scaling factor, and $M_d(t)$ is the base transformation for S_d. We then solve for the new joint angles with the new trajectory $p_d^G(t)$ using the same IK method from previous section. For motion blending, where we have different foot trajectory from each example motion, we compute the weighted sum of the foot positions using the same weights obtained from motion parameterization and solve IK with the weighted foot positions. This ensures that the foot plants in the resulting motions will be intact if there is no foot sliding in the example motions.

In addition, the online constraint enforcements could be used to further modify the motions to adapt for the environments. For example, when walking on a terrain with irregular heights, we can adjust the height of foot position to prevent the foot from penetrating the ground. This can be performed by annotating the example motions to indicate the time interval t_{plant} where the foot is planted on the ground, and the time interval t_{flight} where the leg is in flight. The new foot trajectory can then be warped based on the ground height and the timing information. The key is that the foot positions are free to changed during the flight period, but must be fixed at the correct position during the foot plant period. Let t^s indicates the time when the leg start the flight stage and t^e indicates the end of flight. Because we have the information of original motions, the foot positions can be inferred from the motion at both t^s and t^e. Thus, we know the terrain height h^s at t^s and can also predict the terrain height h^e when the foot is planted at time t^e. During the flight stage, we interpolate the terrain height to offset the constraint position for the moving foot. Specifically, the new foot position will be, as follows:

$$p_c(t) = p_d^G(t) + h(t) \quad (3)$$

$$h(t) = h^s + \frac{t - t^s}{t^e - t^s}(h^e - h^s) \quad (4)$$

The character root position is also offset in a similar manner in y-axis. We need to consider $h(t)$ from both left and right feet, because the new root position would affect the pose for both legs. We pick the smaller height $min(h_{left}(t), h_{right}(t))$ of the two feet to adjust the root position, because it is preferable to bend the leg than to overstretch it. Because $h(t)$ for both feet change continuously, the IK constraints for both feet and root position are also guaranteed not to abruptly jump. Thus, the resulting motion after IK adjustment will maintain continuity without popping.

5.3. Self-collision Handling

Currently, we do not handle collisions caused by the retargeting process. Therefore, there may be intersections between limbs and body when the character performs certain motions or gestures. The main difficulty of run-time collision handling is to efficiently detect self-intersections. Although it is not trivial to perform self-intersection detection in real-time for deformable mesh, we can approximate the results by defining a smooth bounding volumes such as capsules for each bone segment. The intersections between bounding capsules can be found efficiently, and the offsets to resolve collisions would be smooth. The collision resolution would adjust the joint angles based on the offsets to prevent bone segments from intersections. We also need to consider additional issues so the method can simultaneously maintain the IK constraint and preserve the continuity for resulting motions after collision resolution. Designing an effective collision handling method that address the former issues for our online retargeting process is left as future work.

5.4. Performance Trade-off

Online retargeting moves some of the computation from preprocessing to run-time. The performance penalty is an important factor for the applicability of the method. Our method does not incur significant overhead and is able to achieve real-time performance for retargeting tens of characters on-the-fly. Specifically, the motion transfer stage requires only two additional quaternion multiplications per joint, and has very little impact on the overall performance. On the other hand, the constraint enforcement is more demanding and requires about 1 ms per update because it needs to solve for IK. In the future, we would hope to explore faster methods such as the analytical method proposed in [25] for solving IK and further improve the performance.

6. DISCUSSION

6.1. Character Capabilities

The system is able to infuse characters with a number of capabilities, based on a set of controllers primarily driven through various procedurally-based algorithms, as well as through a set of motion examples that are blended together so as to provide a range of behavior and movement. The gazing [21], head movements [21] and saccades [22,26] have been described in previous work and are based on controllers that rely upon joint placement and models of human movement, whereas object manipulation, locomotion and constraints [23], and other primarily parameterized motion data is based on blending similar motion clips together, whose methods have been described elsewhere. Interactive control is primarily carried out via BML [16], a high-level XML interface that allows the specification and coordination of various behaviors together.

It is important to note that low-fidelity motion can be generated without the need for retargeting or the need to identify a full humanoid skeleton, such as generated by [15]. For example, a footstep-based locomotion method can be used in combination with IK to generate basic character movement, and various IK methods can be used to generate reaching and touching actions. However, such movements would lack the fidelity that can potentially be achieved by using high-quality motion examples, and would only be suitable for low-resolution models or characters. By contrast, we offer a pipeline where extremely high-fidelity motion, such as those generated from motion capture, can be incorporated onto high-resolution models and characters.

6.2. Behavior Libraries

We have identified a set of behaviors that enable a virtual character to perform a large number of common tasks such as walking (see Figure 6), gazing, gesturing, touching (Figure 7), and so forth. In the authors opinion, this set represents a minimal, but expressive set of capabilities for a 3D character for traditional uses in games, simulations, and other off-line uses. Behaviors suited for particular environment or specific situation can be added by including and retargeting animation clips, or parameterized sets of similar motion clips parameterized for performances along a range of motion, such as the punching set in Figure 8. However, the focus of this work is to quickly and easily generate a 3D character that would be useful in a wide variety of circumstances, thus the authors feel that a critical aspect to this work is the recognition and inclusion of such behavior sets as part of such a system. We list the behaviors associated with this system in Table I later along with some details of their implementation:

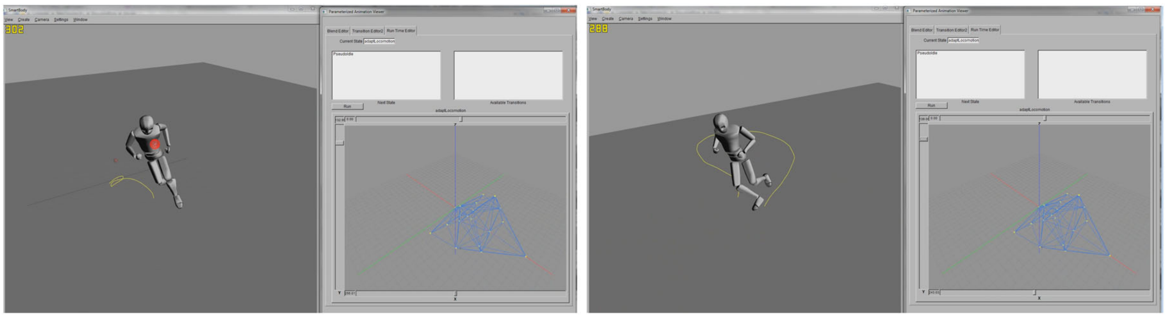


Figure 6. In the figures previously, we map a set of 20 motion captured locomotion animations to drive an arbitrary character. The motion captured locomotion data set is of much higher visual quality than can be generated via procedural techniques such as through the use of IK or footstep models.

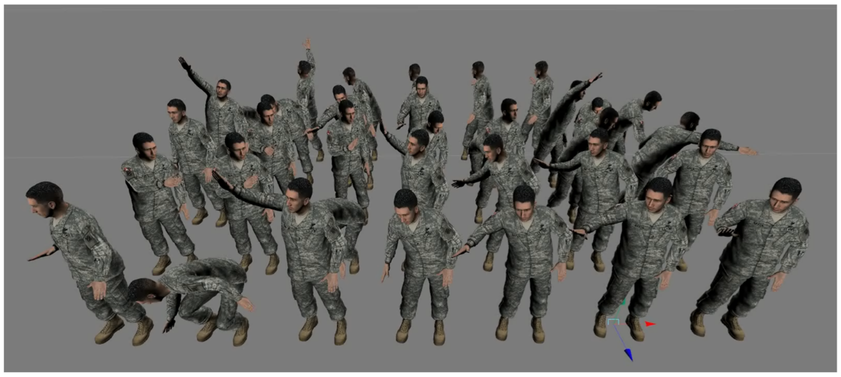


Figure 7. A behavior library for a reaching behavior. Thirty two right-handed reaches and retractions are captured from the same starting pose. The reaches are then mirrored to the left hand. Note that the reaching behavior requires the behavior author to manually annotate the reach phases so as to work properly with grasping, holding, and touching.

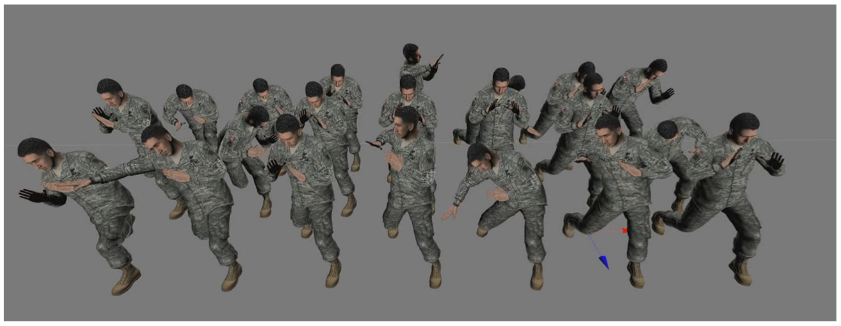


Figure 8. A behavior library for punching behavior. Twenty one right-handed punches are captured from the same starting pose. The punches are then mirrored to the other hand, for a total of 42 motions. Note that this behavior is mostly data-driven, and uses the general blending capabilities of the animation system.

6.3. Towards a Reusable Motion Database

Over the long term, we envision the construction and application of a large number of behavior sets that can be applied to a virtual character for various purposes. For example, a behavior set with a large number and type of fine manipulation tasks could be used for characters demonstrating the use of hand tools. A behavior set for crawling, bicycling, or climbing could be used for the specific environments that use them. Of course, motion alone is not sufficient for rich interaction in a virtual environment. For example, behaviors that are not self-contained and require contact with objects or structures outside of the character itself would necessitate a additional control mechanisms in the animation engine. However, the ability to quickly associate characters with

Table I. Generic behaviors.

Behavior	Description	Comments
Gazing	Look at objects and characters in the scene	Procedural
Head movements	Nodding, shaking, and other head motions	Procedural
Eye saccades	Rapid eye movement	Procedural
Locomotion	Movement and steering around obstacles	Data-driven with procedural elements
Object manipulation	Touching, reaching, grasping, and pointing to objects	Data-driven with procedural elements
Gesturing	Different types of gestures used in conversation	Data-driven with annotations
Jumping	Jump in different directions and lengths	Data-driven

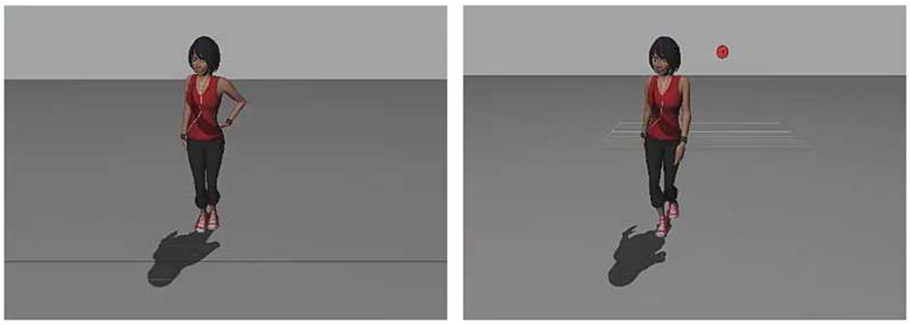


Figure 9. Mapping a customized behavior set onto a character. In this case, a set of locomotion animations stylized for female is mapped onto an arbitrary female character. Note that the choice of behavior sets are chosen by the user at the time of creation.

motion would itself eliminate a bottleneck in the animation pipeline.

By providing an automated means to transfer a set of motions, and potentially, a set of behaviors, onto a character, we envision the widespread development of behavior libraries separate from a specific particular game or simulation. As digital artists create libraries of models for generic use, so too can motion developers capture or design a library of animations for generic use as well. Thus, experts in crafting motion can create both stylized or context-specific motion sets. Game or simulation designers can then choose from a set of motions in the same way that they can choose from a set of models. By loosening the bond between the motion and the model, we greatly increase the use and reuse of digital assets. By contrast, most motion libraries offered are specific to particular characters, specific simulation environments, or represent stand-alone motion clips instead of a broad range of similar useful multipurpose motion.

6.4. Stylizing Behavior Sets

It is important to note that there are wide variations in style among behaviors. For example, walking style can vary greatly between people. Thus, although a locomotion behavior can be automatically infused into a character, all such characters will end up walking in a similar way. This limitation can be remedied in part by providing additional stylized behavior sets (e.g., both male and female locomotion sets, as in Figure 9). Additional variations in style, emotion or performance (such as joyful vs. sad movements) would also require additional behavior sets. Alternatively, the integration of motion style editing or modification research such as those found in [27–33] provide an excellent complement to the incorporation of behavior sets. Such style editing could be applied to an entire behavior set, resulting in a wide variation of performance. For behaviors primarily generated through controllers (such as gaze and nodding), certain settings can be modified to change the style or performance. For example, the speed/intensity at which the gazing is engaged, or the number of repletion for head nods.

6.5. Limitations

We introduced a two-step approach, which applies joint name mapping, motion retargeting, and constraint enforcement to perform online behavior transferring for different virtual characters. There could be difficult cases where the two skeletons have very distinct bone lengths. In such situation, our retargeting may not perform well on the new character.

Our skeleton guessing algorithm is limited to humanoid or mostly humanoid forms. It assumes that characters have human-like structure; two arms, two legs, elbows, shoulders, knees, and so forth. A skeleton with different topology, such as the one shown in Figure 10, would cause incorrect joint mapping results. In order to address this issue, we develop a user interface shown in Figure 11 as part of our behavior transfer pipeline to let user adjust the joint name mapping manually. This semiautomatic approach ensures that our system can still be applicable to

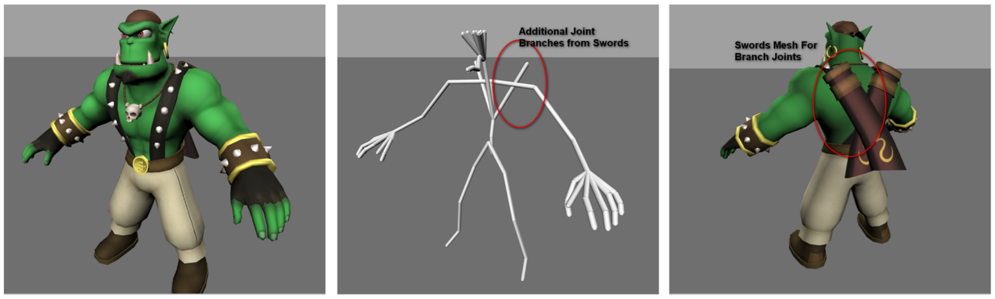
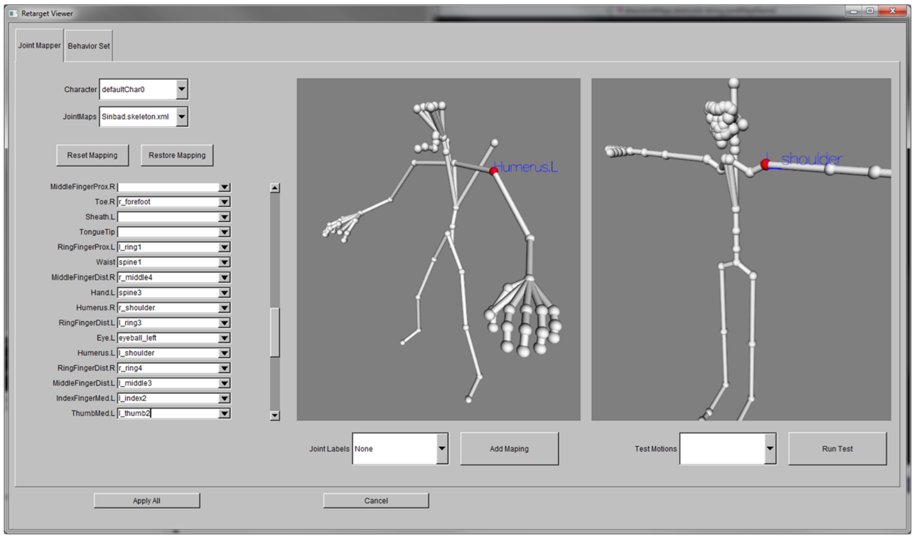

Additional Joint Branches from Swords
Swords Mesh For Branch Joints

Figure 10. The skeleton has additional branches on the spine joints due to the swords on his back, which makes it difficult to determine the arm chains based on skeletal topology. Our joint name guessing method would fail to find correct joint mapping in this case.



Character: defaultChar0
JointMaps: Sinbad.skeleton.xml

MiddleFingerProx.R	
Toe.R	r_forefoot
Sheath.L	
TongueTip	
RingFingerProx.L	l_ring1
Waist	spine1
MiddleFingerDist.R	r_middle4
Hand.L	spine3
Humerus.R	r_shoulder
RingFingerDist.L	l_ring3
Eye.L	eyeball_left
Humerus.L	l_shoulder
RingFingerDist.R	r_ring4
MiddleFingerDist.L	l_middle3
IndexFingerMed.L	l_index2
ThumbMed.L	l_thumb2

Joint Labels: None

Figure 11. The user interface for our behavior transfer procedures. Here, the user can manually correct the results from our joint name guessing method to resolve the issues for certain custom skeletons.

custom skeletons that do not have human-like structures. In addition, many controller-based behaviors require a minimum configuration of joints in order to be fully-realized. For example, the gaze control requires a number of joints, stretching from the lower back to the eyes in order to gaze while engaging several body parts at once. Also, the behavior sets that rely on single or parameterized motion sets require a reasonable match between the original motion subject on which the data was captured and the targeted skeleton. If the skeleton topology or bone proportions fall too far outside of normal human limits, the appearance quality of the behavior will be deteriorated.

Currently, our system does not handle the motion transfers between quadrupedal skeletons. This limitation is mainly due to the joint mapping stage because our algorithm is based on heuristics. Because quadrupedal animals may have different set of joint names, such as paws instead of hands, the existing rules and naming conventions in our system may not resolve the joint mapping correctly. Moreover, some rules need to be revised to accommodate for different topological structure because most animals would have additional tail joints. Thus our method would need new rules to correctly identify joint branches for arms, legs, head, and tail. This could be performed by using additional joint name conventions for tail joints to differentiate between tail and head. Once the joint name mapping is handled correctly, the online retargeting can be trivially extended for quadrupedal skeletons, because the method does not rely on a specific joint hierarchy.

Facial animation and lip syncing is an important part of many games and simulations involving animated characters. However, whereas the topology and hierarchy of skeleton bodies are somewhat standardized, facial topology and hierarchies are not. For example, it is reasonable to assume that a humanoid character has knees, but unreasonable to assume that the same skeleton has a cheek joint. The issue is further complicated by the common use of both blend-shape and joint-based facial animation methods. As a result, little can be assumed about the face of an arbitrary humanoid skeleton to allow the incorporation into an automated pipeline. On the other hand, our system is able to automatically generate both facial expressions and lip syncing to characters who have specified a minimal set of Facial Action Coding System (FACS) units and a small number of mouth shapes used for lip syncing, while incorporating synthesized speech via a text-to-speech engine. Such specification requires the manual creation of those FACS poses and mouth poses. Although such efforts would not take a professional artist very long to create, perhaps requiring only a few hours, these additional efforts lie outside of the automatic pipeline described in this paper.

7. CONCLUSION

We have described a pipeline for incorporating high-quality humanoid assets into a virtual character and quickly infuse that character with a broad set of behaviors that are common to many games and simulations. We believe that by automating the incorporation of models, we are lowering the barrier to entry for end users and potentially increasing the number and complexity of simulations that can be generated.

We offer our entire code base for inspection and evaluation under Lesser General Public License licensing at http://smartbody.ict.usc.edu/.

REFERENCES

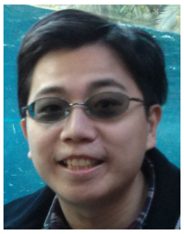
1. Feng A, Huang Y, Xu Y, Shapiro A. Automating the Transfer of a Generic Set of Behaviors onto a Virtual Character. In *Motion in Games*. Springer Berlin Heidelberg, 2012; 134–145.
2. Miller C, Arikan O, Fussell D. Frankenrigs: building character rigs from multiple sources. *IEEE Transactions on Visualization and Computer Graphics* 2011; **17**(8): 1060–1070.
3. Cross-platform game engine with authoring tool, new feature demo of version 4.0 pre-release. http://www.unity3d.com.
4. Arikan O, Ikemoto L. Animeeple character animation tool, 2011.
5. Gleicher M. Retargetting motion to new characters. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98. ACM, New York, NY, USA, 1998; 33–42.
6. Kulpa R, Multon F, Arnaldi B. Morphology-independent representation of motions for interactive human-like animation. *Computer Graphics Forum, Eurographics 2005 Special Issue* 2005; **24**: 343–352.
7. Lee J, Shin SY. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999; 39–48.
8. Monzani JS, Baerlocher P, Boulic R, Thalmann D. Using an intermediate skeleton and inverse kinematics for motion retargeting. *Computer Graphics Forum* 2000; **19**(3). citeseer.nj.nec.com/monzani00using.html.
9. Ho ESL, Komura T, Tai CL. Spatial relationship preserving character motion adaptation. *ACM Transactions on Graphics* 2010; **29**(4): 33:1–33:8. http://doi.acm.org/10.1145/1778765.1778770.
10. Zordan VB, Van Der Horst NC. Mapping optical motion capture data to skeletal motion using a physical model. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2003; 245–250.
11. Shin HJ, Lee J, Shin SY, Gleicher M. Computer puppetry: an importance-based approach. *ACM Transactions on Graphics* 2001; **20**(2): 67–94. http://doi.acm.org/10.1145/502122.502123.
12. Jin Choi K, Seok Ko H. On-line motion retargetting. *Journal of Visualization and Computer Animation* 1999; **11**: 223–235.
13. Glardon P, Boulic R, Thalmann D. Robust on-line adaptive footplant detection and enforcement for locomotion. *The Visual Computer* 2006; **22**(3): 194–209.
14. Kovar L, Schreiner J, Gleicher M. Footskate cleanup for motion capture editing. In *Proceedings of the ACM SIGGRAPH Symposium on Computer Animation*. ACM Press, San Antonio, Texas, 2002; 97–104.
15. Hecker C, Raabe B, Enslow RW, DeWeese J, Maynard J, van Prooijen K. Real-time motion retargeting to highly varied user-created morphologies. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08. ACM, New York, NY, USA, 2008; 27:1–27:11. http://doi.acm.org/10.1145/1399504.1360626.
16. Kopp S, Krenn B, Marsella S, Marshall A, Pelachaud C, Pirker H, Thrisson K, Vilhjálmsson H. Towards a common framework for multimodal generation: the behavior markup language. In *Intelligent Virtual Agents*, Vol. 4133, Gratch J, Young M, Aylett R, Ballin D, Olivier P (eds), Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Heidelberg, Germany, 2006; 205–217.
17. Heloir A, Kipp M. EMBR: a realtime animation engine for interactive embodied agents. In *Intelligent Virtual*

Agents, Vol. 5773, Ruttkay Z, Kipp M, Nijholt A, Vilhjálmsson H (eds), Lecture Notes in Computer Science. Springer Berlin / Heidelberg, Heidelberg, Germany, 2009; 393–404.
18. van Welbergen H, Reidsma D, Ruttkay Z, Zwiers J. Elckerlyc. *Journal on Multimodal User Interfaces* 2009; **3**: 271–284.
19. Cassell J, Vilhjálmsson HH, Bickmore T. Beat: the behavior expression animation toolkit. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01. ACM, New York, NY, USA, 2001; 477–486. http://doi.acm.org/10.1145/383259.383315.
20. Niewiadomski R, Bevacqua E, Mancini M, Pelachaud C. Greta: an interactive expressive ECA system. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '09. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2009; 1399–1400. http://dl.acm.org/citation.cfm?id=1558109.1558314.
21. Thiebaux M, Marsella S, Marshall AN, Kallmann M. Smartbody: behavior realization for embodied conversational agents. In *7th Int'l Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. International Foundation for Autonomous Agents and Multiagent Systems, 2008; 151–158.
22. Shapiro A. Building a character animation system. In *Proceedings of the Fourth International Conference on Motion in Games*. Springer, Berlin, 2011; pp. 98–109.
23. Feng AW, Xu Y, Shapiro A. An example-based motion synthesis technique for locomotion and object manipulation. In *I3D*. ACM SIGGRAPH Symposium on Interactive 3D Graphics, Costa Mesa, CA, March 9–11, 2012; 95–102.
24. Autodesk motionbuilder real-time 3D character animation software. http://www.autodesk.com/motionbuilder.
25. Kallmann M. Analytical inverse kinematics with body posture control. *Computer Animation and Virtual Worlds (CAVW)* 2008; **19**(2): 79–91.
26. Lee SP, Badler JB, Badler NI. Eyes alive. *ACM Transactions on Graphics* 2002; **21**: 637–644. http://doi.acm.org/10.1145/566654.566629.
27. Shapiro A, Cao Y, Faloutsos P. Style components. In *Proceedings of Graphics Interface 2006*, GI '06. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 2006; 33–39. http://dl.acm.org/citation.cfm?id=1143079.1143086.
28. Min J, Liu H, Chai J. Synthesis and editing of personalized stylistic human motion. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '10. ACM, New York, NY, USA, 2010; 39–46.
29. Neff M, Kim Y. Interactive editing of motion style using drives and correlations. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09. ACM, New York, NY, USA, 2009; 103–112.
30. Wang JM, Fleet DJ, Hertzmann A. Multifactor Gaussian process models for style-content separation. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07. ACM, New York, NY, USA, 2007; 975–982.
31. Rose C, Cohen M, Bodenheimer B. Verbs and adverbs: multidimensional motion interpolation. *Computer Graphics and Applications, IEEE* 1998; **18**(5): 32–40.
32. Amaya K, Bruderlin A, Calvert T. Emotion from motion. In *Proceedings of the Conference on Graphics Interface '96*, GI '96. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 1996; 222–229.
33. Hsu E, Pulli K, Popović J. Style translation for human motion. *ACM Transactions on Graphics* 2005; **24**(3): 1082–1089.

SUPPORTING INFORMATION

Supporting information may be found in the online version of this article.

AUTHORS' BIOGRAPHIES



Andrew Feng is currently a research associate in Institute for Creative Technologies. He received the PhD and MS degree in computer science from University of Illinois at Urbana-Champaign. His research interests include character animation, mesh deformation, mesh skinning, and real-time rendering.

Yazhou Huang received his PhD from the University of California Merced in 2012, with emphasis on motion capture based character animation. During this time, he has worked as a graduate research assistant in Prof. Marcelo Kallmann's Graphics Lab, and in summer 2012, he was a visiting research assistant at the USC Institute for Creative Technologies. His research interests include motion parametrization, motion planning, human-computer interaction and robotics. Currently, he is lead R&D engineer at EON Reality, Inc.

Yuyu Xu received her Master degree in May 2010, from University of Southern California. She is currently a research programmer at Institute for Creative Technologies USC, focusing on embodied agent animation and simulation, high level AI that generates nonverbal behaviors for agents.

Ari Shapiro is a research scientist at USC Institute for Creative Technologies where he leads the Character Animation and Simulation research group. For several years, he worked on character animation tools and algorithms at visual effects and video games companies such as Industrial Light and Magic, LucasArts, and Rhythm & Hues Studios. He completed his PhD in computer science at UCLA in 2007 in the area of computer graphics with a dissertation on character animation using motion capture, physics, and machine learning. He also holds an MS in computer science from UCLA, and a BA in computer science from the University of California, Santa Cruz.