In Proceedings of the 5th International Conference on Motion in Games (MIG), Rennes, France, 2012

# Automating the Transfer of a Generic Set of Behaviors Onto a Virtual Character

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

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

**Abstract.** Humanoid 3D models can be easily acquired through various sources, including online. 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. 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.

**Keywords:** animation, graphics, system, retargeting

## 1 Motivation

3D characters are commonly seen in video games, feature films, mobile phone applications and web sites. 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 skillset. For example, 3D models are generated by digital modelers or through hardware-based acquisition, while animators create or apply motion to the characters.

Thus, while 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.

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 [19]. At the time of this writing, [2] 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 [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, but offline retargeting system to generate animations appropriate for a particular skeleton. Retargeting has been an area of much research in the animation community since Gleicher [9]'s work which uses optimization and low-pass filtering to retarget motion. Many other retargeting algorithms use various approaches: Kulpa [16] retargets motion by using angular trajectories, and then solve several body areas, Less[17] uses a hierarchical approach to retargeting, Mozani[21] uses an intermediate skeleton and IK to handle retargeting between skeletons with different topologies. Kulpa[16] retargets motion through a morphology-independent representation by using angular trajectories, and then solving several body areas. Taku[12] uses

spatial relationships for motion adaptation which can handle many contact-based motion retargeting problems. Zordan[31] retargets optical data directly onto a skeleton via a dynamics-based method. Shin[27] uses an online retargeting method via an analytical IK method that prefers the preservation of end effector values. Choi[6] uses a Jacobian-based IK method for online retargeting.

Our retargeting system attempts to find footplants in order to better retarget movement. An online footplant detection and enforcement method is presented in Glardon's work[8]. By contrast our retargeting method enforces footplants offline, and doesn't modify the length of limbs as in [15] so as to be compatible with many game and simulation skeleton formats. Similar to our goals, the work in [10] 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 inverse kinematics in performing online retargeting based on semantic descriptions of movement. By contrast, we are interested in offline, but relatively fast retargeting of high quality motions onto humanoid characters that cannot be achieved via simple walk cycles and reaching constraints. [19] 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) [14]. 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 [11], Elckerlyc [30], Beat [5] and Greta [23] in addition to our system, SmartBody [28, 25]. However, to our knowledege, no other BML-based systems besides our own have implemented extensive character locomotion capabilities or generic capabilities such as object manipulation [7] which are associated with large sets of behaviors. Since the BML specification emphasizes speech, head movements and gestures, most BML-compatible systems emphasize only those features.

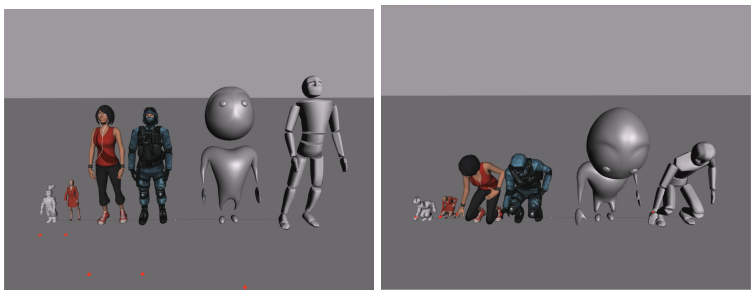

**Fig. 1.** A set of characters from many different sources are automatically retargeted and registered into our system. The characters can now perform a number of tests with controllers and parameterized motions in order to insure that the behavior has been properly transferred: gazing, object manipulation, locomotion, head nodding, etc.

**Fig. 2.** 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.

## 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 like retargeting, and although being a trivial task, it is commonly done by hand. 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, MotionBuilder), results shown in Fig. 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. Fig 3 shows the final mapping result to be achieved from left side mapped to the right side. Left side as an example follows MotionBuilder[1] standard skeleton joint naming convention, and 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 like gaze, reaching and locomotion.

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. Fig 4 (top left) generalizes some of the variations found in our testing skeletons, and the routine is partially 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 source skeleton will have similar properties.

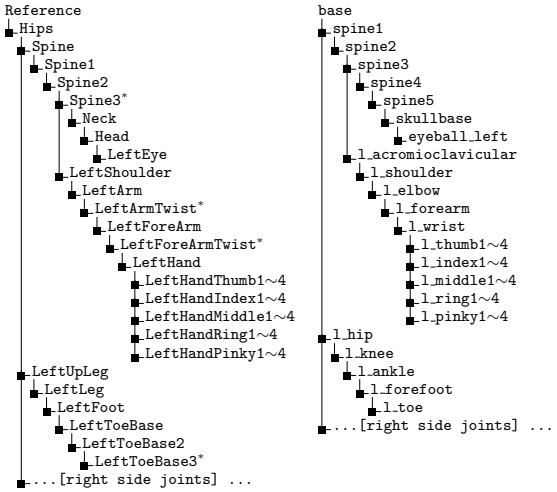


**Fig. 3.** Final mapping result achieved by our system, left side is given as an example following MotionBuilder naming convention, right side is the corresponding joint names in our system. * denotes joint (if exists) is skipped as it's not handled by our system.

Here we could only show a small portion of this procedure, Fig 4 and Algorithm 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 5) for details of the mapping procedure. Characters with uncommon hierarchy/joint names may break the heuristics, such as with the presence of extra joints (wings, tails, etc) or asymmetrical hierarchy, in which case the user needs to manually complete the mapping starting with the partial mapping table generated by the algorithm.

## 4 Retargeting

The motion retargeting process works by transferring an example motion set from our canonical skeleton to a custom skeleton provided by the user. The

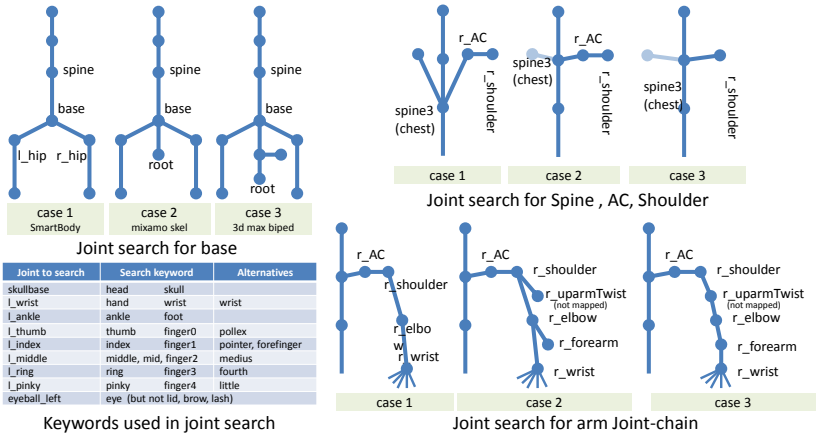


**Fig. 4.** An illustration of various configurations generalized from testing skeletons for certain key joints and joint-chain mapping using heuristics.

**Algorithm 3** Search routine for arm joint-chain.

```
1.  J ← acromioclavicular (AC)
2.  while J ← J.child() do
3.      if J has 5 children then
4.          MAP(wrist, J)
5.      else if J.num_children() = 0 then
6.          J ← 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. J1 ← shoulder ; J2 ← wrist
14. switch (J2.depth − J1.depth)
15. case 2:
16.     uparm ← 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
```

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 and our canonical skeleton. This is done by recursively rotating each bone segment in target skeleton to match the global

**Algorithm 1** Search routine for base joint.

```
1. while i ≤ max_search_depth do
2.     J ← 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
```

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

```
1. J ← base
2. while J ← J.child() do
3.     if J.num_children() ≥ 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
```

direction of that segment in source skeleton at default pose (Fig 5 left) so that the target skeleton is adjusted to have the same default pose as the source skeleton.

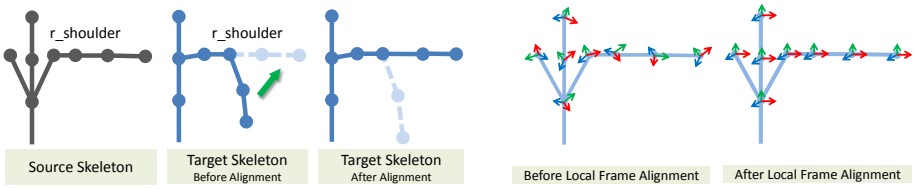


**Fig. 5.** Left side shows alignment of a bone segment between two skeletons so that target skeleton matches the pose of source skeleton. Right side shows re-orientation of joint local frames so that they align with the canonical world frame, which enables straightforward transfer of motion data from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

Once the default pose is matched, we address the discrepancy between their local frames by adding suitable pre-rotation and post-rotation at each joint in target skeleton. Specifically, given a joint $b_i$, with its global rotation $R^G$ and initial local rotation $q^{init}$ when in default T-pose, we re-orient its local frame as $q' = q^{init}\ {R^G}^{-1}\ q\ R^G$ , where $q'$ is the actual local rotation after re-orientation, and $q$ is the standard rotation that complies with the default global frame. In other words, the original local frame of $b_i$ is re-oriented to align with the default canonical global frame as shown in Fig 5 right, e.g. 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 $b_i$. Since 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 done 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 non-canonical alignments, we reverse the re-orientation process as $q = R^G \; {q^{init}}^{-1} \; q' \; {R^G}^{-1}$ 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 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, etc. 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 inverse kinematics to enforce the foot plant constraint in the transferred motions. The inverse kinematic 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 above IK method deforms an input pose to satisfy the end effector constraint, while maintaining the target pose $\mathbf{z}$ as much as possible. 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.

Previous methods exist for detecting and fixing foot sliding [15, 8]. 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. Since 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 $p_s(t), p_d(t)$ be the 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,

$$\begin{aligned} p'_d(0) &= p_d(0) \\ p'_d(t + \delta t) &= p'_d(t) + s_r(p_s(t + \delta t) - p_s(t)) \end{aligned}$$

, where $p'_d$ is the new target trajectory, and $s_r$ is the scaling factor based on leg length from the previous section. The above equation warps the foot trajectory from original skeleton based on the scale of target skeleton. This was proven to work well during our experiments on various skeletons with different limb lengths and proportions. Good robustness was shown on different motion styles with no additional smoothing needed.

## 5 Discussion

**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 [28], head movements [28], and saccades [18, 25] have been described in previous work and are based on controllers that rely upon joint placement and models of human movement, while object manipulation, locomotion and constraints [7] and other primarily parameterized motion data is based on blending similar motion clips together, whose methods have been described elsewhere. Interactive control is primarily done via the Behavioral Markup Language (BML) [14], 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 [10]. 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.

**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, gazing, gesturing, touching 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 offline uses. Behaviors suited for a 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 (locomotion parametrization shown in Fig. 6). 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.

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 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 standalone motion clips instead of a broad range of similar useful multi-purpose motion.

**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, while 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 (for example both male and female locomotion sets, see Fig. 1). Additional variations in style, emotion or performance (such as joyful v.s. sad movements) would also require additional behavior sets. Alternatively, the integration of motion style editing or modification research such as those

found in [26], [20], [22], [29], [24], [3], [13] 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 repetitions for head nods.

**Limitations** Our skeleton mapping 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. 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.

Facial animation and lip syncing is an important part of many games and simulations involving animated characters. However, while 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 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 FACS/mouth poses by artists, however these additional efforts lie outside of the automatic pipeline.

**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 LPGL licensing at http://smartbody.ict.usc.edu/. Please see our accompanying video at : http://people.ict.usc.edu/∼shapiro/mig12/paper9/.

## References

1. Autodesk motionbuilder real-time 3d character animation software, http://www.autodesk.com/motionbuilder
2. Cross-platform game engine with authoring tool, new feature demo of version 4.0 pre-release., http://www.unity3d.com
3. Amaya, K., Bruderlin, A., Calvert, T.: Emotion from motion. In: Proceedings of the conference on Graphics interface '96. pp. 222–229. GI '96, Canadian Information Processing Society, Toronto, Ont., Canada, Canada (1996)

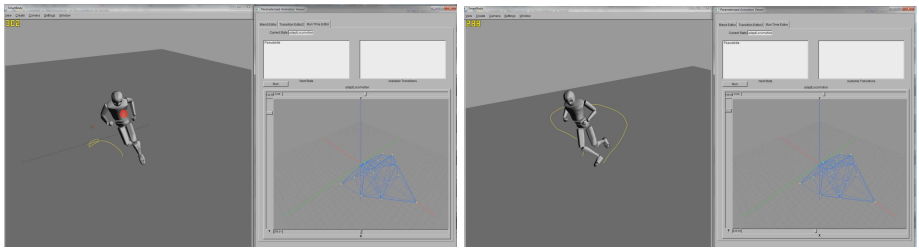

**Fig. 6.** In the figures above, we map a set of 20 motion captured locomotion animations to drive an arbitrary character. The motion captured locomotion data set is of much higher visual quality than can be generated via procedural techniques such as through the use of IK or footstep models.


4. Arikan, O., Ikemoto, L.: Animeeple character animation tool (2011)
5. Cassell, J., Vilhjálmsson, H.H., Bickmore, T.: Beat: the behavior expression animation toolkit. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. pp. 477–486. SIGGRAPH '01, ACM, New York, NY, USA (2001), http://doi.acm.org/10.1145/383259.383315
6. jin Choi, K., seok Ko, H.: On-line motion retargetting. Journal of Visualization and Computer Animation 11, 223–235 (1999)
7. Feng, A.W., Xu, Y., Shapiro, A.: An example-based motion synthesis technique for locomotion and object manipulation. In: I3D. pp. 95–102 (2012)
8. Glardon, P., Boulic, R., Thalmann, D.: Robust on-line adaptive footplant detection and enforcement for locomotion. Vis. Comput. 22(3), 194–209 (Mar 2006)
9. Gleicher, M.: Retargetting motion to new characters. In: Proceedings of the 25th annual conference on Computer graphics and interactive techniques. pp. 33–42. SIGGRAPH '98, ACM, New York, NY, USA (1998)
10. Hecker, C., Raabe, B., Enslow, R.W., DeWeese, J., Maynard, J., van Prooijen, K.: Real-time motion retargeting to highly varied user-created morphologies. In: ACM SIGGRAPH 2008 papers. pp. 27:1–27:11. SIGGRAPH '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1399504.1360626
11. Heloir, A., Kipp, M.: Embr a realtime animation engine for interactive embodied agents. In: Ruttkay, Z., Kipp, M., Nijholt, A., Vilhjlmsson, H. (eds.) Intelligent Virtual Agents, Lecture Notes in Computer Science, vol. 5773, pp. 393–404. Springer Berlin / Heidelberg (2009)
12. Ho, E.S.L., Komura, T., Tai, C.L.: Spatial relationship preserving character motion adaptation. ACM Trans. Graph. 29(4), 33:1–33:8 (Jul 2010), http://doi.acm.org/10.1145/1778765.1778770
13. Hsu, E., Pulli, K., Popović, J.: Style translation for human motion. ACM Trans. Graph. 24(3), 1082–1089 (Jul 2005)
14. Kopp, S., Krenn, B., Marsella, S., Marshall, A., Pelachaud, C., Pirker, H., Thrisson, K., Vilhjlmsson, H.: Towards a common framework for multimodal generation: The behavior markup language. In: Gratch, J., Young, M., Aylett, R., Ballin, D., Olivier, P. (eds.) Intelligent Virtual Agents, Lecture Notes in Computer Science, vol. 4133, pp. 205–217. Springer Berlin / Heidelberg (2006)
15. Kovar, L., Schreiner, J., Gleicher, M.: Footskate cleanup for motion capture editing. In: Proceedings of the ACM SIGGRAPH Symposium on Computer Animation. pp. 97–104. ACM Press, San Antonio, Texas (2002)

16. Kulpa, R., Multon, F., Arnaldi, B.: Morphology-independent representation of motions for interactive human-like animation. Computer Graphics Forum, Eurographics 2005 special issue 24, 343–352 (2005)
17. Lee, J., Shin, S.Y.: A hierarchical approach to interactive motion editing for human-like figures. In: Proceedings of the 26th annual conference on Computer graphics and interactive techniques. pp. 39–48. SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1999)
18. Lee, S.P., Badler, J.B., Badler, N.I.: Eyes alive. ACM Trans. Graph. 21, 637–644 (July 2002), http://doi.acm.org/10.1145/566654.566629
19. Miller, C., Arikan, O., Fussell, D.: Frankenrigs: Building character rigs from multiple sources. Visualization and Computer Graphics, IEEE Transactions on 17(8), 1060 –1070 (aug 2011)
20. 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. pp. 39–46. I3D '10, ACM, New York, NY, USA (2010)
21. Monzani, J.S., Baerlocher, P., Boulic, R., Thalmann, D.: Using an intermediate skeleton and inverse kinematics for motion retargeting. Computer Graphics Forum 19(3) (2000), citeseer.nj.nec.com/monzani00using.html
22. 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. pp. 103–112. SCA '09, ACM, New York, NY, USA (2009)
23. 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. pp. 1399–1400. AAMAS '09, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2009), http://dl.acm.org/citation.cfm?id=1558109.1558314
24. Rose, C., Cohen, M., Bodenheimer, B.: Verbs and adverbs: multidimensional motion interpolation. Computer Graphics and Applications, IEEE 18(5), 32 –40 (sep/oct 1998)
25. Shapiro, A.: Building a character animation system. In: Proceedings of the Fourth International Conference on Motion In Games. Springer, Berlin (2011)
26. Shapiro, A., Cao, Y., Faloutsos, P.: Style components. In: Proceedings of Graphics Interface 2006. pp. 33–39. GI '06, Canadian Information Processing Society, Toronto, Ont., Canada, Canada (2006), http://dl.acm.org/citation.cfm?id=1143079.1143086
27. Shin, H.J., Lee, J., Shin, S.Y., Gleicher, M.: Computer puppetry: An importance-based approach. ACM Trans. Graph. 20(2), 67–94 (Apr 2001), http://doi.acm.org/10.1145/502122.502123
28. Thiebaux, M., Marsella, S., Marshall, A.N., Kallmann, M.: Smartbody: behavior realization for embodied conversational agents. In: 7th int'l joint conference on Autonomous agents and multiagent systems (AAMAS). pp. 151–158. International Foundation for Autonomous Agents and Multiagent Systems (2008)
29. Wang, J.M., Fleet, D.J., Hertzmann, A.: Multifactor gaussian process models for style-content separation. In: Proceedings of the 24th international conference on Machine learning. pp. 975–982. ICML '07, ACM, New York, NY, USA (2007)
30. van Welbergen, H., Reidsma, D., Ruttkay, Z., Zwiers, J.: Elckerlyc. Journal on Multimodal User Interfaces 3, 271–284 (2009)
31. Zordan, V.B., Van Der Horst, N.C.: Mapping optical motion capture data to skeletal motion using a physical model. In: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation. pp. 245–250. SCA '03, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2003)