Building a Character Animation System

Ari Shapiro*

Institute for Creative Technologies
shapiro@ict.usc.edu
http://www.arishapiro.com

Abstract. We describe a system for animating virtual characters that encompasses many important aspects of character modeling for simulations and games. These include locomotion, facial animation, speech synthesis, reaching/grabbing, and various automated non-verbal behaviors, such as nodding, gesturing and eye saccades. Our system implements aspects of character animation from the research community that yield high levels of realism and control.

Keywords: animation, character, graphics, system

1 Motivation

Animating virtual humans is a complex task. Many different aspects of human behavior needs to be modeled in order to generate a convincing result. The behavior and appearance of a virtual characters needs to be recognizably human in expression, although photorealism is not necessary. People are adept at recognizing movement and human-like behavior, so the actions and appearance of a virtual character must match the expectations of the human viewer. This means that not only must the character's movements be natural, but they must be contextually appropriate, such as responding with the appropriate reaction and in the proper time frame to stimuli. Research has been done on various aspects of character animation, such as locomotion and facial animation. However, the integration of all these aspects leads to complexities. For example, coordinating locomotion with path finding, or coordinating reaching with gazing. At first glance, it appears that modeling an entire animated character can be achieved by combining individual areas, and then reassembling the final character as a combination of each individual part. For example, locomotion can be combined with a lip sync animation. This combination works since there is little relationship between locomotion and the movement of a character's lips and face. Serious problems arrive when the areas overlap and directory or indirectly impact each other. For example, performing a manipulation with your hands while simultaneously looking at another object in the virtual world. The looking behavior might engage parts of the character's body that disrupt the manipulation. Thus, although manipulation and gazing are distinct problem areas in animation research, they can interact with each other in unexpected ways.

* shapiro@ict.usc.edu

1.1 Goals

We aim to synthesize a highly realistic, interactive character, which will likely require high-quality and possibly expensive methods. Many game engines provide robust solutions to many real time simulation problems such as mesh rendering, lighting, particle effects and so forth. However, game engines generally do not handle complex character animation. They often provide a general framework for replaying animations on a hierarchical skeleton, as well as a providing mechanism for blending between animations or looping an animation. However, the intricate and specific motion commonly associated with humans must be constructed by the game engine programmer and designer. One of the goals of this project is to develop a character animation system allows the realization of common behaviors that are used in real time games and simulations. These behaviors include: synthesizing speech, responding to speech, moving, touching, grabbing, gesturing, gazing, breathing, emotional expression and other non-verbal behavior.

The system is not intended to be a framework for the development of character animation via a well-defined interface or with pluggable animation blocks. Such well-defined interfaces are effective for well-defined and understood problems. However, animating a virtual character to a high level of realism has a number of complexities that are not well understood, and thus don't benefit greatly from such simple architectures. Such designs can either under specify the interface, leaving too much work to the game designers and programmers, or overly simplifying the system, restricting it's capabilities.

2 System Summary

The animation system [10] is designed around a hierarchical, controller-based architecture [3]. The state of the character is manipulated by series of controllers, with the output of one passed as the input to another. Each controller can either override, modify or ignore the state of the virtual character. The controllers know the state of the character during the last step, as well as the state of the character during the evaluation phase. The controller stack, which controls the state data flow, is listed in Table 2 in the order of execution.

Order	*Controller*	*Comments*
1	World offset	Global orientation and position
2	Idle motion	Underlying idle pose
3	Locomotion	Overrides idle pose during locomotion phases, ignored during idle states
4	Animation	Non-locomotive animations, can encompass entire body or just upper body during locomotion.
5	Reach	Allows reaching and pointing using arms
6	Grab	Hand control for touching, grabbing, and picking up objects
7	Gaze	Looking with eyes, head, shoulders and waist
8	Breathing	Chest and diaphragm control, mostly independent from rest of skeleton hierarchy
9	Constraint	Allows constraints that may have been violated due to impact of preceding controllers (i.e. keeping character's hands on a table while turning to look at another object
10	Eye saccades	Fast movements for eyes, blended with results from gaze
11	Blink	Periodic blinking control
12	Head	Controls head movements; nods, shakes, tilts, backchanneling
13	Face	Determines activation for blend shapes or bone activations when using joint-driven faces, excluding eyes
14	General parameters	Generic controller for transferring non-skeleton data to the rendering engine, such as blushing, tears, GPU shader values, etc.
15	Override	Allows overriding of state values. Useful when taking control of character from other input devices, such as the Kinect

2.1 Problems with Generalization/Specialization Hierarchy

Some controllers can hide the impact of earlier controllers by overriding the state values. For example, the face controllers (13) overwrites the face state originally generated by the idle motion controller (2). This scheme will work for these two controllers, since the face control can be thought of as a specialization of the more generic idle pose. However, the locomotion controller (3) must entirely replace the effects of the idle motion controller (2) during a certain behaviors, such as walking. Thus, while the hierarchy implies a generalization-specialization scheme, in practice, many controllers have effects that overlaps, extend or replace the effects of earlier one. As another example, the gaze controller can engage the entire spine during a gaze behavior when orienting a character's entire body towards a gaze target. However, this will disrupt an animation of, say, a character

whose hands have been placed on a table by the animation controller (4). To restore these implicit constraints, the constraint controller (9) is activated to reposition the character's hands according to the constraints.

This controller scheme for managing character state requires many additional rules to effectively control the entire character. Also, certain controllers, such as the blink and saccade controllers (10 and 11) can work independently from the face (13), thus strict ordering is not necessary. Better models need to be developed for controlling character state that more closely match the interaction of different character behaviors with each other.

2.2 Platforms

The system is written almost entirely in C++, and has been ported to run on both Linux, OsX. At the time of this writing, we are in the process of porting to the mobile platforms, Android and iOs. The system is licensed under LGPL and is available for download at: http://sourceforge.net/projects/smartbody/

2.3 Locomotion

We have implemented and experimented with two different locomotion systems; a semi-procedural system based on [2], and an example-based system.

The semi-procedural locomotion algorithm uses two example motions; a forward walk and a strafe (sideways walk). The procedural nature of the algorithm allows the use of inverse kinematics to place the feet at the desired position as well as control the angle of the foot on uneven terrain. Since only two example motions are used algorithm, the motion can be parameterized along two axes, each representing either forward or sideways movement. Turning is controlled by setting step targets and orienting the body in line with the footsteps.

The drawbacks to using such this semi-procedural system is the lack of ability of the algorithm to allow for differing styles of movement. Because the foot placement is established by using inverse kinematics, the nuances of the leg movement from the animation data are replaced with the results from the IK algorithm. Thus, many different styles of walking on different characters tend to look very similar below the character's waist. In addition, the use of foot steps makes the motion appear to be clomping, or hard stepping, as compared with motion captured or hand-animated motion.

The example-based locomotion shown in Figure 2 includes 19 different animations to control forward, turning and lateral movement. This locomotion engine does not use IK, and relies almost entirely on blending the motion data, notwithstanding the offset of the character in world space. The example-based locomotion currently uses 5 animations for different speeds of forward movement, 5 different animations for turning left at various speeds, 5 animations for turning right at different speeds. The forward movement animations consist of two walking or running cycles, and the turning animations consist of a character turning around at various speeds; turning in place, turning in a tight circle while walking, turning in a tight circle while running and so forth. The size of the turning

circle limits the amount that a character can turn while moving at a given velocity. The animations are parameterized in three dimensions; forward velocity, turning angle, and sideways velocity. Thus, it is possible to emulate any number of foot positions through a combination the various parametric animations. The parameter space of these dimensions can be modeled using tetrahedrons. The parameter values are automatically extracted from the example motions, such as determining the average forward velocity of a motion.

We have found that the example-based animation produces a more realistic result, although this method can be susceptible to foot skating if the motion data is not consistent.

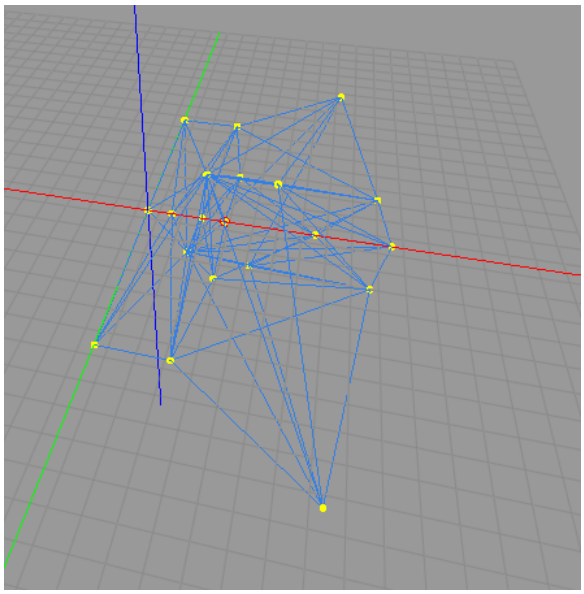

Fig. 1. Visualization of the example-based locomotion. Yellow dots on the red axis indicate moving forward (walking, jogging, running, etc.) Yellow dots along the green axis indicate turning motions. Yellow dots along the blue axis represent strafing, or sideways movement.

2.4 Path Finding

The system uses SteerSuite [8] to handle path finding. The separation of locomotion from path finding allows for the development of each area separately. For example, we have connected the SteerSuite path finding to one of three locomotion engines; the semi-procedural and example-based system as described in the section above, as well as to a simple offset engine that alters the global position of the character without changing it's pose. The simple engine is used to test the

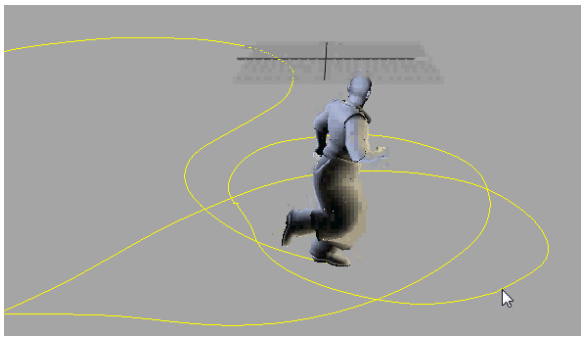

Fig. 2. Example-based locomotion and path.

viability of the path planning engine. The more complex engines produce more realistic-looking motion.

The benefits of separating the two problem areas has a consequence; without knowing the limits of locomotion capabilities, the path planner sometimes requires movements that are sometimes unrealistically fast, and sometimes visually unappealing. For example, the path planner might decide to suddenly switch direction to avoid an obstacle faster than the locomotion system can realistically move the character, thus causing a discrepancy between the planned path and the actual path.

We noticed that many path planners handle large scale movement, such as traversing long distances are large objects, and very few path planners handle intricate movement in small areas and indoors.

2.5 Reaching and Grabbing

Our system utilizes an example-based approach for reaching. From a standing position, we use 32 different reaching motions to allow a character to most objects within arms-length. Each arm uses 16 example motions from four elevations (high, mid-high, mid-low, low). Motions created for one hand can be mirrored to the other hand in order to reduce the number of examples needed. We interpolate the reach examples to generate pseudo-examples [4], as shown in Figure 3. Our reaching algorithm first finds closest sets of examples, and interpolates and time warps the motion to produce the final motion. Inverse kinematics is then used to achieve the exact desired location, similar to [1].

Because of this reaching technique is based on examples, a different example set is required for each type of reaching motion. For example, a series of reaching tasks performed while sitting requires a different set than the standing reaching set, as in Figure 3. Reaching during walking or running would require another example set. Although it is also possible to overlay the reaching examples on the upper body while animating the lower body, this will cause a loss of realism, since the body will not preserve its movement dynamics. Currently, this reaching

system does not handle collisions; it assumes a clear path of movement for the reaching. In addition, the reaching examples are synthesized from the starting position, to the target, then back to the original position. The ability to reach one target, then switch to another without returning to the original pose is outside of the capabilities of the original examples. For such cases, we blend between the pose targets, resulting in passable, but not high quality, reaching animation between targets.

For grabbing, we use heuristic that determines the orientation of the hand that is needed to grab an object. For example, the grab controller will rotate the hand in order to reach around the thinnest axis of the target object. A long, narrow object will cause the hand to reorient so that the long axis is parallel to the palm, allowing the fingers and thumb to close around the object. Near end of reach behavior, the hand will blend between the current hand pose and a specific grabbing pose, which can be modified for larger or smaller objects. Collision spheres are placed on the fingers that allow detection of contact between them and the target object. Individual fingers are represented as IK chains, and will stop blending between the original hand pose and the grabbing hand pose once contact is detected with the object. Each finger IK chain reacts to the collisions independently. Thus, the hand can be seen to wrap around the target object.

The synthesize reaching and grabbing motion produces a more convincing result when appropriately timed gazing is added to the motion. The virtual character can be timed to look at the object before reaching is started, and maintained through the grabbing motion.

2.6 Facial Animation and Speech Synthesis

The focus of our animation system is to develop autonomous characters that can think, react and perform various tasks in a dynamically changing environment. If the characters needed only to duplicate an actor's performance, the greatest amount of realism would come from performance capture coupled with prerecorded audio. However, since the dialogue of our characters is not always known in advance, nor is the content, it is important to generate a model that can produce arbitrary utterances with reasonable-looking facial movements. Our system uses a set of visemes that are activated by a text-to-speech engine (TTS). The TTS engine translates an utterance in text format into a series of phonemes (word sounds) and time markers. These phonemes and then mapped to a smaller set of visemes (a facial movement that matches a word sound) which are used to drive the facial animation. Originally, our system used a simple scheme by creating a one-to-one mapping between phonemes and visemes. Each phoneme would trigger its corresponding viseme, and be phased-in and phased-out by overlapping the phase-out period of one viseme with the phase-in period of a second viseme. However, the visual result of many phonemes can be overpowered by the visual result of other phonemes, such as the combining an 'r' with an 'o'. Thus, many phonemes can effectively be ignored when animating the face. In addition, many phonemes produce similar-looking visemes, so those duplicates can be represented by a single viseme. In addition, the system incorporates a set

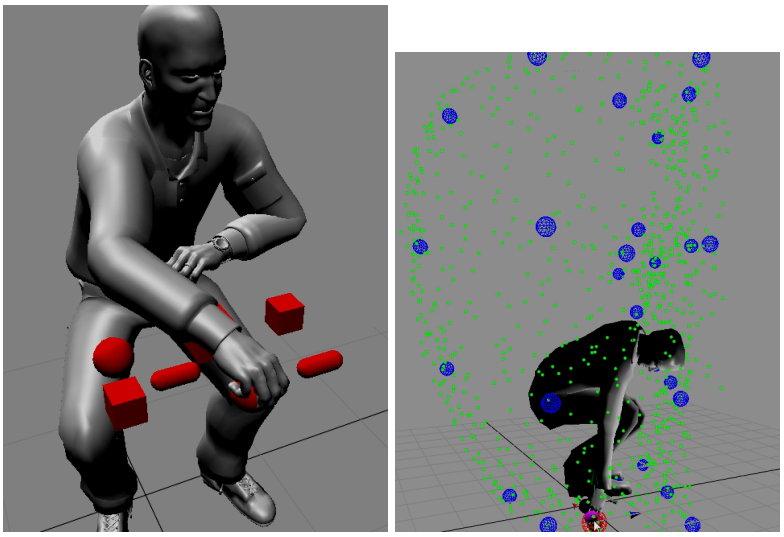

Fig. 3. (Left) Reaching and grabbing during sitting. Notice that the pinky of the right hand did not collide with the target object, and thus was allowed to blend into the grabbing pose, whereas the other fingers collided with the target object, and remain at the collision surface. (Right) Example-based reaching for right side of body. The blue spheres represent the original examples, while the green dots are examples interpolated from the original examples. Note that the character engages his entire body during the reaching motion.

of the Facial Action Coding System (FACS) units, which can be used to express emotion and display facial movements unrelated to speech.

Speech synthesis is implemented using a number of text-to-speech systems. The system can also replay prerecorded audio, when first preprocessed with viseme and timing information.

2.7 Modeling Eye Movements and Saccades

Eye movement is an important part of conveying emotion and intent when animating digital characters. Humans frequently shift their focus of attention among and between objects in the environment, as well as performing eye saccade motions to reduce cognitive load. However, many simulations involving characters ignore eye movements, yielding a virtual character that stares blankly for long periods of time, a behavior not seen in healthy humans.

The eyelid controller regulates the position of the upper and lower eyelids in relation to the pitch of the eyeball. Thus causing the eyelid to move up or down relative to the character's eye gaze direction. The system adds a small delay to the eyelid tracking speed, in order to visually separate the effect of the different physical structures of the eyeball and the eyelid. The impact can be seen in Figure 4.

For eye movement, we implement an eye saccade model based on [7], which uses a statistical eye movement model for listening and speaking. In many cases, the effect of using the saccade model is effective, as it simulates the more complex cognitive processes of the character. However, these statistical models do not account for the differing context of the speech that is being heard, nor of the content of the speech being uttered. This can sometimes cause a discrepancy between the speech being uttered and eye movements associated with that speech, which can lead to a loss of realism, or even propel the characters into the Uncanny Valley. In addition, these statistical models do not consider the factors that drive the eye movements, such as objects of interest to the character, or movements in the visual field that would cause a spark of attention.

To develop a greater sense of realism, we seek to develop an eye movement model for animated characters based on a study of humans interacting with an animated character Figure5. Participants listen to a digital character speak and respond to simple questions about personal information, as well as perform and mental tasks, such as counting backwards. An eye tracker captures the location of the eye fixation on the screen containing the animated character. In addition, a video of the participant responding to the questions is synchronized with the eye tracker data for reference. The eye fixation data is then analyzed separately according to whether the participent is listening to, speaking to or reacting to the animated character. The questions are designed to elicit different kinds of emotional reactions: boredom, surprise, mental load, recall of simple information, and so forth. With this model, we hope to generate a more complex and contextually appropriate set of eye saccade behaviors.

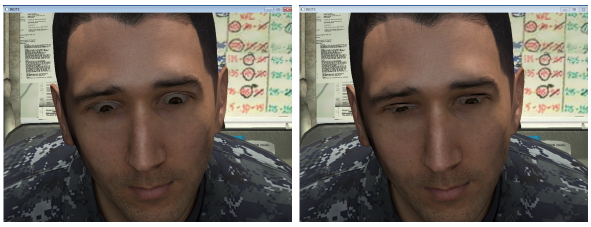

Fig. 4. Lowering the lid to the level of the eye. This is achieved by activating the upper lids in combination with the eye pitch. Although the amount of this effect is small, the impact on the viewer is large, as the character appears to be in a completely different emotional state in the two images.

2.8 Head Nods, Head Shakes, and Gazing

Our system has full control of head movements, which can reflect nodding, shaking and various other head movements. Gazing can occur from a hierarchy of joints, including the eyes, neck, shoulders and waist [9]. A number of parameters can be used to alter the gazing as well as head nodding and shaking styles, such

Fig. 5. Visualization of eye movement during listening. The size of the bubble indicates the length of time spent fixated on the point or area, while the lines indicate the path travelled. Note that the listener briefly looked at the animated character's left hand during a gesture, while remaining mostly fixated on the mouth.

as speed, offset, timing and so forth. In practice, most users of these capabilities tend to vary the default movements only slightly with timing, and generally do not alter the movement style. This is likely due to the complexity of the parameters, and that realistic human head and gazing movement cannot easily be specified using such parameters. In the future, we would like to pursue an example-based approach to head movements and gazing.

2.9 Breathing

Breathing is controlled my retiming and looping a breathing animation cycle that functions on a set of joints that model the chest and abdomen of the virtual character. Although more complex physical models exist [11], such effects can be simulated kinematically since the effects are often hidden underneath clothing. The breathing controller can also activate certain facial animations, such as the flaring of nostrils and the opening of the mouth.

2.10 BML Realizer

The system uses the Behavioral Markup Language (BML) [5] as an interface for controlling and synchronization speech, gesturing and other aspects of conversational modeling. Since the Vienna Draft of BML was originally designed for conversational agents, it heavily emphasizes such behaviors, with little or no support for aspects of character animation such as locomotion, reaching and so forth. Thus, our system enhances the BML with a number of extensions to support this functionality.

2.11 Non-Verbal Behavior

Designing a character animation system requires the development of a number of different capabilities for a virtual character. However, the decision to use those

capabilities much be left to the simulation designer, game designer, or agent (AI) developer. For example, deciding to walk towards a particular object resides in the domain of the agent, whereas navigating around obstacles should clearly be part of the motion engine. Some character capabilities fall in between the clearly defined areas of motion and intention. Aspects such as gesturing during an utterance or head movement to model awareness of the environment are such examples. They should be configurable by a designer, but represent unconscious, natural or repetitive behavior that lies close to the motion-level behavior.

Our solution to this problem is to employ a separate component that handles non-verbal behavior such as gesturing, head nodding, idle gaze behavior and facial expression, based on [6]. This component enhances or changes instructions from the agent before sending them to the motion engine. Thus, a complex motion that engages many different behaviors at once, such as head nodding while blinking, speaking and emoting, can be hidden from the agent designer. This intermediate component will extract syntactical and semantic information from an utterance, and add additional gestures, head nods, or blinks to the motion instruction. In addition, this component can trigger idle behaviors, such as consulting a saliency map that dictates an attention model to allow natural idle behavior during lull periods of a simulation.

3 Conclusion

One of the goals of our research is to locate, implement and improve techniques that can produce high quality animation. To that end, we have experimented with example-based techniques for both locomotion and reaching that yield high-quality results. We intend to replace other aspects of our system, such as head nods, shakes and gazing with a similar example-based approach. The tradeoff for achieving this level of quality comes at the expense of generating a large number of example motions for differing scenarios, which can be both slow and expensive, and remains an obstacle to development.

Acknowledgments. The SmartBody software is a combined effort of dozens of people over many years. Thanks to the current and past SmartBody team members, including Marcus Thiebaux, Yuyu Xu and Wei-Wen Feng for their hard work on developing and maintaining the system. Also thanks to the Integrated Virtual Humans team including Ed Fast, Arno Hartholt, Shridhar Ravikumar, Apar Suri, Adam Reilly and Matt Liewer for their integration, testing and development efforts. Most importantly, thanks to Stacy Marsella who was the inspiration behind SmartBody and who continues to help shape its development.

References

1. Camporesi, C., Huang, Y., Kallmann, M.: Interactive motion modeling and parameterization by direct demonstration. In: Proceedings of the 10th International Conference on Intelligent Virtual Agents (IVA) (2010)

2. Johansen, R.S.: Automated Semi-Procedural Animation for Character Locomotion. Master's thesis, Aarhus University, the Netherlands (2009)
3. Kallmann, M., Marsella, S.: Hierarchical motion controllers for real-time autonomous virtual humans. In: Proceedings of the 5th International working conference on Intelligent Virtual Agents (IVA'05). pp. 243–265. Kos, Greece (September 12-14 2005)
4. Kovar, L., Gleicher, M.: Automated extraction and parameterization of motions in large data sets. In: ACM SIGGRAPH 2004 Papers. pp. 559–568. SIGGRAPH '04, ACM, New York, NY, USA (2004), http://doi.acm.org/10.1145/1186562.1015760
5. Krenn, B., Marsella, S., Marshall, A.N., Pirker, H., Thrisson, K.R., Vilhjlmsson, H.: Towards a common framework for multimodal generation in ecas: The behavior markup language. In: In Proceedings of the 6th International Conference on Intelligent Virtual Agents, Marina. pp. 21–23 (2006)
6. Lee, J., Wang, Z., Marsella, S.: Evaluating models of speaker head nods for virtual agents. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1. pp. 1257–1264. AAMAS '10, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2010), http://dl.acm.org/citation.cfm?id=1838206.1838370
7. 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
8. Singh, S., Kapadia, M., Faloutsos, P., Reinman, G.: An open framework for developing, evaluating, and sharing steering algorithms. In: Proceedings of the 2nd International Workshop on Motion in Games. pp. 158–169. MIG '09, Springer-Verlag, Berlin, Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-10347-6_15
9. Thiebaux, M., Lance, B., Marsella, S.: Real-time expressive gaze animation for virtual humans. In: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1. pp. 321–328. AAMAS '09, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2009), http://dl.acm.org/citation.cfm?id=1558013.1558057
10. Thiebaux, M., Marshall, A., Marsella, S., Kallmann, M.: Smartbody: Behavior realization for embodied conversational agents. In: Seventh International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS) (2008)
11. Zordan, V.B., Celly, B., Chiu, B., DiLorenzo, P.C.: Breathe easy: model and control of human respiration for computer animation. Graph. Models 68, 113–132 (March 2006), http://dl.acm.org/citation.cfm?id=1140961.1140965