In *Proceedings of the 5th International Working Conference on Intelligent Virtual Agents (IVA)*, Kos, Greece, September 12-14, 2005, 243-265

Hierarchical Motion Controllers for Real-Time Autonomous Virtual Humans

Marcelo Kallmann[1] and Stacy Marsella[2]

[1] Institute for Creative Technologies[**]
University of Southern California
kallmann@usc.edu
[2] Information Sciences Institute
University of Southern California
marsella@isi.edu

Abstract. Continuous and synchronized whole-body motions are essential for achieving believable autonomous virtual humans in interactive applications.
We present a new motion control architecture based on generic controllers that can be hierarchically interconnected and reused in real-time. The hierarchical organization implies that leaf controllers are motion generators while the other nodes are connectors, performing operations such as interpolation, blending, and precise scheduling of children controllers. We also describe how the system can correctly handle the synchronization of gestures with speech in order to achieve believable conversational characters. For that purpose, different types of controllers implement a generic model of the different phases of a gesture.

1 Introduction

Interactive virtual humans [1] are software artifacts that look and act like humans but are embedded in a virtual world where they interact with humans much like humans interact with each other. To cohabit a virtual world with a human, a virtual human needs to perform a range of behaviors. It must be able to look and move around its environment, pick-up objects and engage in conversation with a human.

To support human-like interactions, these behaviors must be performed continuously in realistic and meaningful ways. Specifically, we want these behaviors to play a similar role in the virtual human's interaction with humans as they do in human-human interaction. For example, people's nonverbal behaviors perform a variety of roles during conversational interactions. Gestures can emphasize or qualify what is being said. They can also substitute for words, for example by conveying greetings, goodbyes, insults, spatial relations or physical properties of objects. They also convey attitudes and reactions to events. The manner of gestures in particular reveals affective information. Gestures also serve to regulate

[**] The first author is now affiliated with the University of California, Merced

dialog interaction. For example, a speaker can relinquish a dialog turn with a gesture that metaphorically offers the turn to the next speaker, using an upward facing palm, or can seize the turn with a vertical hand with palm facing the speaker, essentially a “stopping” gesture. This expressiveness is not unique to gestures. Gaze behavior plays a similarly rich set of roles in human interactions.

There are several basic challenges that must be addressed in creating a virtual human body that can realize such behaviors. Because the virtual human is interacting with a human within a virtual world, it must be prepared to react to unexpected events either created by the human or by the virtual world itself. As such, body motions must be interruptable. Similarly, it must be prepared to adjust to the physical constraints implied by the virtual world itself. The virtual human may also be doing more than one thing at a time, for example, walking, gesturing and talking at the same time.

Further, virtual human’s behaviors typically involve multiple parts of the body being in motion. Visually realistic gaze behavior, for example, often requires the careful coordination of a range of motions, including eye movements, head/neck movements, twisting of the joints in the torso as well whole-body stepping movements as necessary. Furthermore, gaze may also have to be coordinated and synchronized with other body movements and communication channels, e.g. with the phonemes and visemes of speech.

It is our experience that these requirements need behavioral flexibility. The virtual human body must be prepared to compose and synchronize multiple behaviors both sequentially and simultaneously, in a continuous fashion that allows it to be embedded and interacting with humans in the virtual world. Alternatives, such as long duration carefully crafted full body motions may look better but over-reliance on them restricts the behavioral responses of the virtual human and causes breakdowns in interactions with humans.

Indeed, a blend of animation approaches is required to achieve both flexibility and realism. The ability to point and look at an unexpected event arbitrarily located in the virtual world is a key capability in a virtual human. Procedural approaches for realizing this capability (e.g. using Inverse Kinematics [2]) are flexible and can exhibit sufficient realism, especially given that the human’s attention is divided between the virtual human and the object that is being attended to. On the other hand, revealing a dejected affective state by the virtual human looking downward might be more effectively realized by crafted animations or motion captured sequences [3].

Because of the important role the virtual human’s “physical” behavior plays as well as the challenges posed in realizing those behaviors, a key aspect of any virtual human is the animation algorithms and their coordination that constitute its “body”. To address these concerns, we propose a motion control architecture based on generic controllers that can be hierarchically interconnected in real-time in order to achieve continuous motion respecting given constraints.

The approach is inspired by neuroscience evidence that complex motor behavior might be obtained through the combination of motor primitives [4]. Primitive controllers in our system are motion generators that can be built with arbitrary animation algorithms, such as keyframe interpolation or procedural animation. Primitive controllers can then be connected to higher level controllers performing different kinds of operations, such as scheduling, blending and interpolation.

In particular we show how our system handles the synchronization of motion segments with speech. The problem is formulated as a scheduling problem where motion controllers are sequenced and blended with gaps filled by interpolation, according to given timing constraints.

2 Related Work

A wide range of computer animation techniques have been proposed in the literature [2]. In particular for interactive virtual humans, different motion generation techniques are available: walking [5] [6] [7], reaching and object manipulation [8] [9] [10], Inverse Kinematics [11] [12] and keyframe interpolation of designed or motion captured [3] keyframes.

We focus on this work on the integration of such different animation techniques in a single animation platform. This involves building abstractions encapsulating the output of motion controllers which can then be blended, resulting in seamless transitions between the different controllers. Such integration has been already employed in some sense in previous systems [13] [14] [15]. In particular, the *AgentLib* system [15] [16] encapsulates controllers as specialized actions, which can then be blended for achieving smooth transitions. There are some new animation packages available in the web that seem to address similar issues, however no precise information about the employed techniques were found.

Our approach builds on these previous models by adding the capability of hierarchically organizing the motion flow between controllers. This allows the creation of controllers which are in fact modifying and/or controlling children motion controllers. For example, an open-door controller would control two children controllers (walking and reaching) for achieving the needed coordination for opening the door. In the end, the open-door controller is seen as any other motion controller and can be further processed and sequenced with other controllers. In our architecture, complex object interactions [17] can thus be seen as a standard motion controller. This notion of hierarchical organization has been already used in previous systems [18], however in the context of dynamical simulation and not with the goal of synchronized scheduling and blending as in our work.

The architecture is specially well suited for handling the speech-gesture synchronizations required in conversational characters [19] [20] [21]. We show how two controllers (a scheduler and an interpolator) are able to synchronize predesigned gesture motions in order to achieve perfect synchronization with speech.

3 Conversational Characters Requirements

We list here some of the particular requirements for synchronizing motion controllers with speech that have helped motivate our approach.

Meaning of motion The dynamic and spatial qualities of motions convey meaning in different subtle ways. In some cases this quality can most readily be preserved with carefully designed motions, built by skilled artists or via motion capture. On the other hand, some motions such as deictics/pointing can usually be synthesized with different Inverse Kinematics techniques [11] [12]. In our framework, generic motion controllers are used independent of the underlying motion generation technique.

Gesture structure Gestures have a structure comprised of several phases [22]. A preparatory phase brings the hand/arm into position to perform the gesture. The stroke phase is the main part of the gesture and largely carries its meaning. This phase is closely synchronized to the corresponding speech. There is also a relaxation phase where the gesture ends with the hand being in a resting pose. Between those phases, there may also be hold phases where the arm and hand are more or less stationary (see Fig. 1). The appropriate temporal manipulation of these phases is an important means to manipulate the expressive, affective quality of the motion as well as control the synchronization of gesture, speech and social interaction.

Continuous gesturing People regularly perform one gesture right after another. In such cases, the gesture does not go to a fully relaxed position but rather relaxes into a "gesture space" (roughly in front of the speaker), ready to perform another gesture. We call this continuous gesturing. Many factors influence the likelihood of such gesturing, including personality, cultural differences, situational and relational factors, and the arousal of the speaker. We see the ability to do such gestures as an important means to convey these factors. From the standpoint of the animation system, this suggests that there must be some means to compose, and if necessary procedurally adapt, the motions that comprise the gestures in order to achieve fluid, realistic motion over the multiple gestures.

Full body motion The virtual human will only look realistic if the full body moves in a synchronized way. In the case of human gestures, gestures may involve motion of not only the hand and arm, but also the rest of the body. In fact, gestures that only involve the arm and hands often look robotic, or unexpressive. This is especially true of gestures that have strong dynamics and large spatial extent, both of which are an effective ways to convey arousal. It is also critical that the gesture be closely synchronized with the head and gaze motion. Shifts in the temporal relation between gestures and gaze/head movement can alter the interpretation of the gesture.

Synchronization Motions must be able to be synchronized with external events. In the case study considered here, external events are timings for speech synchronization.

4 Motion Engine Architecture

The motion engine architecture is a C++ class library allowing the creation and interconnection of arbitrary skeletons (i.e. characters) and motion controllers.

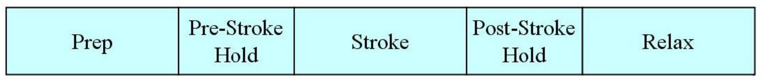


Fig. 1. The considered phases of a gesture.

A controller is a generic self-contained object that maintains the description of the skeleton joints being controlled and as well the respective joint values of the generated motion at a given time. Joint values can then be sent to a skeleton or connected to other controllers for further processing. Generic methods are available for querying and evaluating controllers at any given (monotone) time. The specific motion generation algorithms are not relevant for using the controllers.

4.1 Character Representation

A character is represented as an articulated figure composed of hierarchical joints. Each joint rotation is parameterized differently and contains individually placed joint limits according to its anatomical properties.

The local rotations of joints are always represented in quaternion format [2]. However, different parameterizations can be defined for each joint. For joints with 3 degrees of freedom (DOFs), the swing and twist decomposition is usually preferred [23]. These joints have a local frame with the z-axis lying along the main axis of the corresponding limb. The swing rotation axis is always perpendicular to the z-axis and the twist rotation that follows is simply a rotation around the z-axis. The swing motion is therefore a rotation around a vector lying in the x-y plane and is represented as a 2D axis-angle $\mathbf{s} = (x, y)$, where $\mathbf{s}$ is the rotation axis and $\|\mathbf{s}\|$ is the rotation angle. Such representation allows straightforward use of spherical ellipses for meaningfully bounding the swing motion [23], which is very important for developing procedural controllers, for example based on Inverse Kinematics (IK) [11]. The twist rotation is bounded with minimum and maximum values.

Joints with 2 DOFs are either parameterized with a swing axis-angle, or with 2 Euler angles. For instance, the elbow and knee joints need to be parameterized with flexion and twist Euler angles while the wrist and ankle joints are better parameterized with a swing axis-angle and its ellipsoidal limits. As twist rotations, Euler angles are bounded with minimum and maximum values. Similarly, the remaining joints of the character are parameterized with Euler angles, swings, twists, or with quaternions as appropriate.

4.2 Channels

Each controller specifies the preferred type of parameterization in each controlled joint. For instance, controllers based on keyframe interpolation only need to interpolate joint values and therefore quaternion parametrization with no joint limits is usually the best choice. Procedural controllers however will prefer other parameterizations and with joint limits.

We use the term *channel* to specify one piece of information controlling a joint. Channels can describe one DOF, such as “x-rotation”, or “y-translation”, but they can also describe a 3-DOF rotation with a quaternion or a 2-DOF swing rotation. The used channels are therefore described as an ordered list in the following format:

$$C = ((j_1, c_1), ..., (j_n, c_n)), \tag{1}$$

where j_i is the joint identifier, and c_i is a descriptor of the used parameterization for that joint, $0 \leq i \leq n$. Given the channels description C, a buffer containing joint values for C is denoted as:

$$B_C = (v_1, ..., v_m), \tag{2}$$

where v_j is the j^{th} channel value, $0 \leq j \leq m$, according to the channels description C ($n \leq m$). For example if C describes a linkage with 1 swing rotation, and 1 Euler angle, we will have $m = 3$, which is the number of values required according to C. Note that, as controllers may choose different parameterizations for a same joint, joint values can always be converted to a same quaternion format when needed (e.g. to be blended).

4.3 Connections

Each controller defines the channels C to be used, and has an evaluation function that generates and stores the joint values for a given time t in buffer B_C. Usually t must be evaluated monotonically, but some controllers (as keyframe interpolators) allow evaluation at arbitrary times.

Controllers can be mainly of two types. Source controllers generate motion while connector controllers receive as input motion from source controllers, and perform further operations such as interpolation, blending, filtering, scheduling, etc. Connectors can also be connected to other connectors, achieving arbitrary hierarchical configurations. At any time, any controller can be connected to a skeleton for final visualization of the motion in the character. Fig. 2 exemplifies possible connections. In the figure, controller A affects the joints of the right arm and the resulting arm motion can be blended with another motion coming from controller B; the blending operations are decided by controller C.

Connections require a matching of joint identifiers and parameterization types. Each channels description C maintains a hash table with all pairs (j_i, c_i) (as in equation 1). These pairs are composed of integer identifiers and therefore simple and compact hashing functions can be used. When a controller with buffer B_{C_1} is mapped to another controller with buffer B_{C_2}, each channel in C_1 is searched for a matching channel in C_2. Relying on hash table allows a linear time matching algorithm. After the matching is done, a *mapping list* is obtained and tells exactly where in B_{C_2} each entry in B_{C_1} should be mapped to. Non-mapped channels are detected and can be treated as needed (e.g. ignored). The same mapping process is used to map controllers to skeletons, with the difference

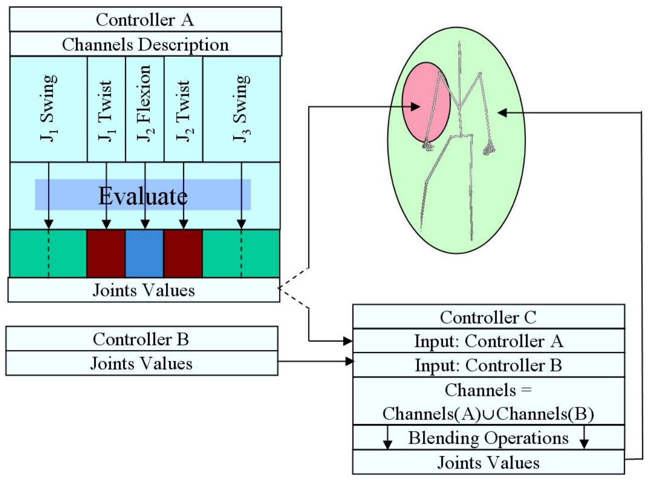


Fig. 2. Example of connections. When controller A is evaluated at a given time, it will fill its buffer with the joint values resultant from the evaluation. Controller A can be connected directly to a skeleton, or alternatively, connected to controller C for a blending operation with controller B.

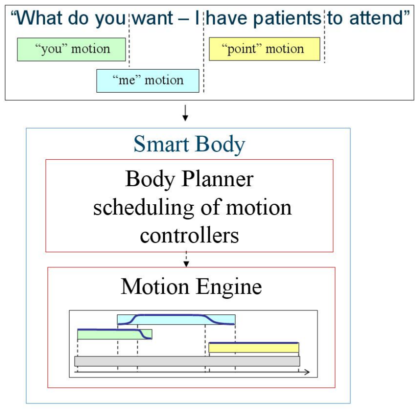
that the mapping is done directly to the joints of the skeleton. Controllers have a generic apply method that will send the current values in their buffer either to the buffer of another mapped controller or to any skeleton attached.

5 Controllers for Scheduling Synchronized Gestures

Besides the fact that motion controllers are self-contained objects, they also contain higher level semantic information about the generated motions. For example parameters such as minimum and maximum allowed time warping factors, point of emphasis, etc, are available and are used for making decisions in the scheduling process.

We focus now on the problem of composing controllers for matching timing constraints for a conversational character. The problem is specified with annotated text telling the exact times when each motion should have its point of emphasis played. The timing annotation is specified with a text-to-speech synthesis system.

Motions are *played* with a keyframe-interpolator controller and synchronized with scheduler and controller-interpolator controllers. Composing these controllers is the role of a *body planner* that determines in real time a schedule of controllers to be played (see Fig. 3). The composition planning process is an extension of existing approaches [20] that takes into account the extra parameters introduced by the system. We present now each of the used controllers.



"What do you want – I have patients to attend"
"you" motion
"point" motion
"me" motion
Smart Body
Body Planner scheduling of motion controllers
Motion Engine

Fig. 3. Controllers are composed and scheduled in real time by a body planner, and the final result satisfying the given timing constraints is *played* by the motion engine.

5.1 Keyframe Interpolator

The implemented keyframe interpolator controller basically interpolates joint values in quaternion format without considering joint limits, as the input key postures are supposed to be correct.

In general, commercial modeling packages store joint rotations with Euler angles, and we have created plug-ins that convert and export them directly in quaternion representation. In that way, the interpolator can directly apply spherical linear interpolation between keys without conversions and few other optimizations are possible, as for instance a quaternion does not require trigonometric functions when converted to a rotation matrix, which is the final representation needed by the graphics cards.

Exported motions are annotated with the local time of their emphasis points, and furthermore contain minimum and maximum time warping factors for allowing different operations when fitting the motions in the final solution schedule. For that purpose, we have developed an interactive application that is able to load the exported motions and annotate them with such parameters (see Section 7).

5.2 Scheduler

The scheduler (Fig. 4) is the main controller used for synchronizing children controllers. It keeps a stack of controllers, which are evaluated at any given time t in an ordered fashion, from the bottom of the stack to the top of the stack.

When inserted in the scheduler, several parameters are available for defining the evaluation behavior of the overall schedule, as for example the blending periods for ease-in and ease-out transitions. When a controller is evaluated, if the evaluation time falls inside a blending interval, the values resulted from the

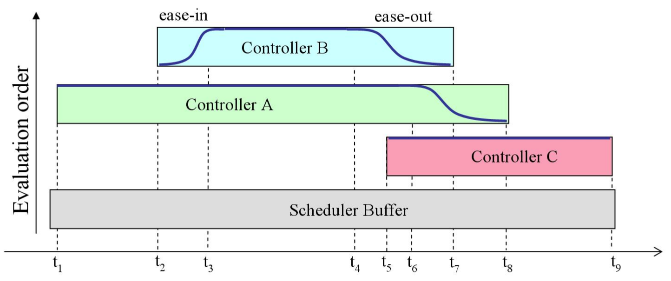


Fig. 4. Scheduler.

evaluation are blended with the current values in the scheduler buffer, and the result is put back in the same buffer. Therefore the final behavior is a pairwise blending sequence from the bottom of the stack to the top of the stack, equivalent to a layering mechanism.

Additional parameters are available, for example to extend a controller duration by repeating its last frame during a desired time interval, after the controller completion. Early commencement of a controller can similarly be obtained.

Note that the scheduler also specifies start and end times to activate controllers and therefore the blending may only occur between controllers which are active at the same given time. In general, blending occurs mainly between pairs of adjacent controllers in the timeline. Even if other behaviors can be achieved, the main purpose of the scheduler is to compose a motion sequence of arbitrary length, with blends producing seamless transitions between controllers. The architecture also allows a same controller to be scheduled several times (even blending with itself), and to schedule other scheduler-controllers, achieving arbitrary hierarchical configurations.

5.3 Interpolator

A specific interpolator-controller was designed for blending the output motions of two children controllers according to a user specified blending curve (see Fig. 5). The blending curve allows designers to explore different kinds of effects and is represented as a piecewise cubic spline editable via control points.

The interpolator can be configured in different ways. It can be used for interpolating between the last frame of one controller and the first frame of the subsequent controller, for example to fill the holds between the gesture phases shown in Fig. 1.

The interpolator can also be configured to interpolate the results of controllers while they are being evaluated. Each time the interpolator is evaluated, it will first evaluate its children controllers and then blend the results in each of the buffers for filling its own buffer. Fig. 5 shows an example where during t_1 and t_2 controller 1 is being evaluated and its result is blended with the first frame of controller 2. During t_2 and t_3 the last frame of controller 1 is blended

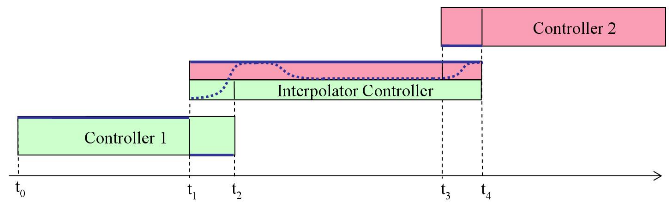


Fig. 5. The interpolator controller blends the results of controller 1 and controller 2 according to a user-specified blending curve.

with the first frame of controller 2, and during t_3 and t_4 the the last frame of controller 1 is blended with the result of the controller 2 evaluation. Note that such interpolator can have an arbitrary length, affecting the time duration between t_2 and t_3. The blending curve can be easily scaled as needed in order to fit arbitrary lengths.

6 Final Gesture Implementation

The gesture model depicted in Fig. 1 is finally implemented by combining the presented controllers. For each given gesture with a time constraint for its point of emphasis (see Fig. 3), the three keyframe-interpolator controllers designed for the gesture's *prep*, *stroke* and *relax* phases are sequenced in a scheduler. They are placed so as to respect the timings of the emphasis points of the *stroke* phases.

However depending on the required times, additional adjustments have to be made. Usually the motions are not long enough to provide a continuous motion, i.e., there might be empty spaces in the schedule timeline. In this case, empty spaces are filled with interpolators, which will produce motion for the "hold phases" (see Fig. 6).

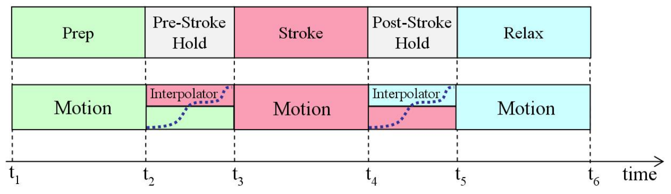


Fig. 6. Scheduling the phases of a gesture with several controllers. Keyframe animation motion controllers can vary their length with local time warping, and interpolator controllers can have arbitrary length. The gesture planner is responsible for finding the best schedule for respecting given timing constraints.

In the case the speech is faster than the disponible motions, there will be no time for holds and the keyframe-interpolator controllers might even be required to be time-warped for producing faster motions respecting the needed timings. In really fast speech cases the controllers of the *prep* and *relax* phases can even be removed from the scheduler. These cases give an idea of the decisions made by the body planner depicted in Fig. 3 and illustrates the several possibilities that can be handled by our system.

Fig. 7 shows some snapshots of our experiments with a conversational doctor character. During these experiments, the presented controllers have shown to be well suited for composing gesture motions under time constraints and at the same time providing a continuous full-body motion.

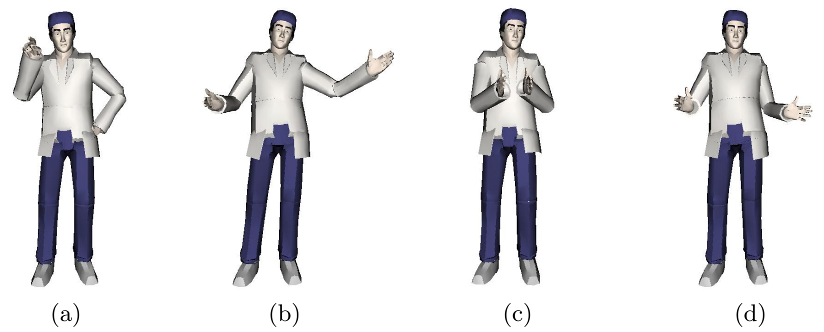


Fig. 7. Several postures from a talking doctor character protecting a clinic: a) "you are the threat...", b) "do you see that patient over there...", c) "we need to protect these people...", d) "what are you waiting for...".

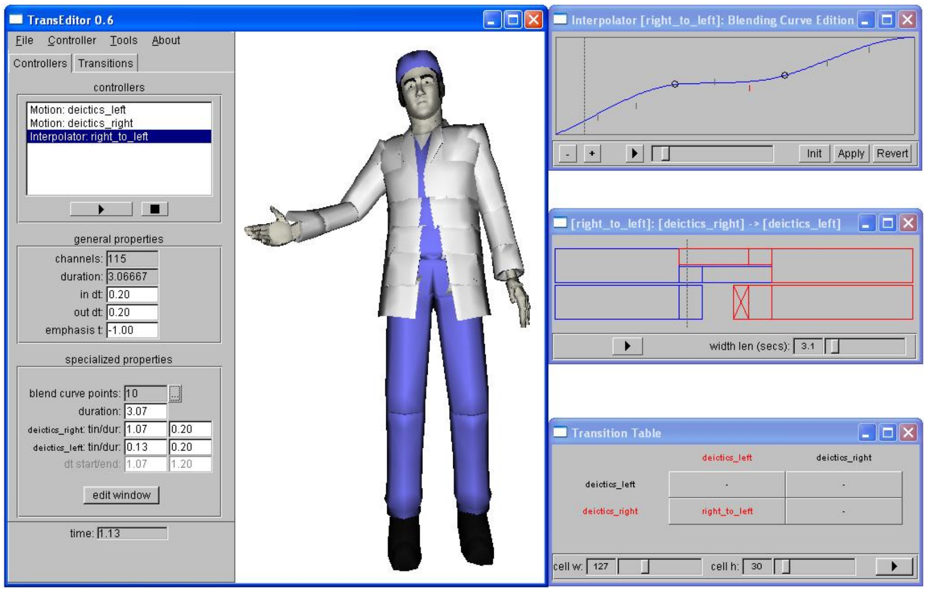
7 Current System

Our current Motion Engine system is under test in a larger project and consists of a stand-alone C++ library and a few applications for editing and testing the created controllers. Fig. 8 shows a snapshot of the main tool used for specifying controllers. We have also written a few Maya mel scripts for exporting motions and characters created by skilled animators.

8 Conclusions

This paper presents an animation system based on the hierarchical organization of generic motion controllers, which can be interconnected and connected to characters arbitrarily in real-time. The approach is similar to the organization of 3D scenes in modern scene graphs, where nodes can be of several types (shapes, engines, transformations, etc).

We believe that such generic organization is of main importance for achieving complex motion for autonomous virtual humans, and we show how controllers can be scheduled in order to synchronize gestures for conversational characters.



controllers

Motion: deictics_left
Motion: deictics_right
Interpolator: right_to_left

general properties

channels: 115
duration: 3.06667
in dt: 0.20
out dt: 0.20
emphasis t: -1.00

specialized properties

blend curve points: 10
duration: 3.07
deictics_right: tin/dur: 1.07 0.20
deictics_left: tin/dur: 0.13 0.20
dt start/end: 1.07 1.20

time: 1.13

width len (secs): 3.1

	deictics_left	deictics_right
deictics_left	-	-
deictics_right	right_to_left	-

cell w: 127 cell h: 30

Fig. 8. Example of some windows used for editing and testing the several parameters of the interpolator controller.

As future work we intend to include additional channels for sensing the environment and for synchronization between concurrent controllers commanding different parts of the body. A challenge would be to develop algorithms for the emergence of complex controllers from given primitive ones [4].

Acknowledgments The project or effort described here has been sponsored by the U.S. Army Research, Development, and Engineering Command (RDECOM). Statements and opinions expressed do not necessarily reflect the position or the policy of the United States Government, and no official endorsement should be inferred.

References

1. Badler, N.I., Phillips, C.B., Webber, B.L.: Simulating Humans: Computer Graphics, Animation and Control. Oxford University Press (1993)
2. Watt, A., Watt, M.: Advanced Animation and Rendering Techniques. ACM Press (1992)
3. Bodenheimer, B., Rose, C., Rosenthal, S., Pella, J.: The process of motion capture: Dealing with the data. In Thalmann, D., van de Panne, M., eds.: Computer Animation and Simulation '97, Springer NY (1997) 3–18 Eurographics Animation Workshop.
4. Thoroughman, K.A., Shadmehr, R.: Learning of action through combination of motor primitives. Nature **407** (2000) 742–747
5. Boulic, R., Magnenat-Thalmann, N., Thalmann, D.: A global human walking model with real-time kinematic personification. The Visual Computer **6** (1990) 344–358

6. Park, S.I., Shin, H.J., Shin, S.Y.: On-line locomotion generation based on motion blending. In: Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA), New York, NY, USA, ACM Press (2002) 105–111
7. Multon, F., France, L., Cani, M.P., Debunne, G.: Computer animation of human walking: a survey. Journal of Visualization and Computer Animation **10** (1999) 39–54
8. Kallmann, M.: Scalable solutions for interactive virtual humans that can manipulate objects. In: Artificial Intelligence and Interactive Digital Entertainment (AIIDE), Marina del Rey, CA (2005)
9. Liu, Y., Badler, N.I.: Real-time reach planning for animated characters using hardware acceleration. In: Proceedings of Computer Animation and Social Agents (CASA'03). (2003) 86–93
10. Kuffner, J.J., Latombe, J.C.: Interactive manipulation planning for animated characters. In: Proceedings of Pacific Graphics'00, Hong Kong (2000) poster paper.
11. Baerlocher, P.: Inverse Kinematics Techniques for the Interactive Posture Control of Articulated Figures. PhD thesis, Swiss Federal Institute of Technology, EPFL (2001) Thesis number 2383.
12. Tolani, D., Badler, N.: Real-time inverse kinematics of the human arm. Presence **5** (1996) 393–401
13. Granieri, J.P., Crabtree, J., Badler, N.I.: Production and playback of human figure motion for visual simulation. ACM Transactions on Modeling and Computer Simulation **5** (1995) 222–241
14. Perlin, K., Goldberg, A.: Improv: A system for scripting interactive actors in virtual worlds. In: Proceedings of SIGGRAPH 96, New Orleans, LA (1996) 205–216
15. Boulic, R., Bécheiraz, P., Emering, L., Thalmann, D.: Integration of motion control techniques for virtual human and avatar real-time animation. In: Proceedings of the ACM International Symposium on Virtual Reality Software and Technology (VRST'97), Switzerland (1997) 111–118
16. Emering, L., Boulic, R., Molet, T., Thalmann, D.: Versatile tuning of humanoid agent activity. Computer Graphics Forum **19** (2000) 231–242
17. Kallmann, M.: Interaction with 3-d objects. In Magnenat-Thalmann, N., Thalmann, D., eds.: Handbook of Virtual Humans. first edn. John Wiley & Sons (2004) 303–322
18. Faloutsos, P., van de Panne, M., Terzopoulos, D.: Composable controllers for physics-based character animation. In: SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, New York, NY, USA, ACM Press (2001) 251–260
19. Kopp, S., Wachsmuth, I.: Synthesizing multimodal utterances for conversational agents. Computer Animation and Virtual Worlds **15** (2004) 39–52
20. Cassell, J., Vilhjálmsson, H.H., Bickmore, T.W.: Beat: the behavior expression animation toolkit. In: Proceedings of SIGGRAPH. (2001) 477–486
21. Carolis, B.D., Pelachaud, C., Poggi, I., de Rosis, F.: Behavior planning for a reflexive agent. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Seattle (2001)
22. McNeill, D.: Hand and mind: What gestures reveal about thought. The university of Chicago Press (1992)
23. Grassia, S.: Practical parameterization of rotations using the exponential map. Journal of Graphics Tools **3** (1998) 29–48