EUROGRAPHICS 2002 / I. Navazo Alvaro and Ph. Slusallek
(Guest Editors)

Dynamic Motion Models

Stefan M. Grünvogel, Thorsten Lange and Jens Piesk

Laboratory for Mixed Realities,
Institute at the Academy of Media Arts Cologne
Am Coloneum 1, 50829 Cologne, Germany
{gruenvogel,lange}@lmr.khm.de

Abstract

Real-time animation of virtual characters is a demanding task. We propose dynamic motion models which can create the motion of a virtual character in different styles and have the ability to change the animation while it is played. The motions are created by blending and changing given base motions which contain animation data and additional information. The blending and manipulation operations on the animation data build up a motion tree which can be altered dynamically in real-time. This allows to change the animation of a motion model while its animation is played. Motion trees of different motion models are blended together to build a motion tree which represents the complete animation of the character. This tree is constructed and dynamically changed according to user commands.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Animation

1. Introduction

Modern avatar technologies and computer games have the need of high level interfaces to create motion data for articulated figures. The idea is to use knowledge about motion to create building blocks of animations which execute certain tasks autonomously. These motion generators have to be adaptable to fulfill different requirements, such as producing a motion in different styles or to take into account certain constraints. A common approach is to take a library of given animation data produced by classical key frame animation or by motion capture and combine them to build more complex animations.

Current implementations differ from their level of abstraction of the motion description and the way they create motions from given animation data. We use the notion *motion model* which was introduced by Grassia [4] (cf. Section 4). Motion models denote parameterized motion generators which create motions fulfilling atomic tasks, e.g. walking, pointing gestures, throwing of things etc. We transfer Grassia's approach to the real-time situation where the animation has to be created on the fly and the output of active motion models can be altered during their execution.

The outline of the paper is at follows. In the next section, we briefly describe related work. Section 3 shows the overall system environment. In Section 4 we explain the concept of motion models and show in Section 5 how we combine different operators on animation data. In Section 6 the procedure for changing dynamically motion trees is explained by an example. The combination of animation coming from different motion models by a controller is shown in Section 7. We conclude with a brief discussion of experimental results, the method's advantages and disadvantages and directions of further work.

2. Previous Work

Badler et.al. [1] specify motions in the Jack System by control parameters which describe biomechanical variables. They also introduce *motion goals*, which are low level tasks their animation system can solve. A similar approach is studied by Hodgins et.al. [6].

Tu and Terzopoulos [15, 16] use biomechanical simulation to animate virtual fish. They have several *Motor Controllers* that carry out specific motions of the fish which are parameterized by the biomechanical parameters.

Within the Improv-System of Perlin and Goldbergs [11] human motions are described and parametrized by so called *Actions*. These *Actions* can be combined by blending them or building transitions between them. Their parameters denote possible perturbations of the original motion data by coherent noise signals. Perlin and Goldberg also state, that it is not always possible to combine every given motion with any other at the same time. For example it makes no sense to combine a *stand* pose with a *walk* motion. Taking this into consideration, they divide *Actions* into different groups, like *Gestures*, *Stances* etc. These groups provide the necessary information about the allowed combinations with other motions.

Sannier et.al. [13] and Kalra et.al. [9] present a real-time animation system VHD which allows users to control the walking of a character with simple commands like *walk faster*.

Grassia[4] introduces the term *motion model*, which we adopt. Motion models represent elementary tasks which can not be divided further. The level of abstraction of the motion models resembles the approach in Perlin et.al[11]. The idea is that every human motion belongs to a certain category e.g *walk*, *run*, *wave with hands*, *throw*, which can stand for itself. Each motion model has its own parameters which controls the process of motion generation.

Having the abstract description of the motions another difficult task is to generate the corresponding motion. The main requirement is to take motion data and combine the animations without losing the special characteristic or flavor of the motion.

Motions of articulated figures are commonly given by a set of curves representing the translational and rotational values of the joints over time. Thus the idea to interpret these curves as signals is nearby. Witkin and Popović [18] introduced the term *Motion Warping*, denoting the transformation of motions of articulated figures in space and in time. Unuma and Takeuchi [17] apply Fourier transformations on motion captured data of human walking. By changing the Fourier coefficients they achieve a change in the characteristic of the original motions while preserving the overall look of the animations. Bruderlin and Williams [2] propose motion multiresolution filtering of animation data. They also introduce methods for changing the temporal and spatial look of the motions. While these last two approaches use Euler angles for the representation of rotational data, Lee [10] introduces multiresolution filtering and analysis of motion data where the rotational values are given by quaternions. For quaternions, simple methods for interpolation between rotations exist [14]. Furthermore, they are nonsingular representations of rotations, needing less memory than rotation matrices. Due to these advantage, we also use this approach.

3. System Architecture

The system architecture follows closely the three layer approach of Perlin et.al.[11]. The *Choreography Editor* creates high level commands from the user input, which are sent to an *Animation Engine*. The *Animation Engine* converts the commands into animation data for a virtual character in real-time. The animation data which consists of translation and rotation data of the joints of the virtual character is sent to the *Render Engine*. The *Render Engine* holds the geometry data of the character, calculates the mesh deformation corresponding to the current joint states of the character and finally generates the graphical output.

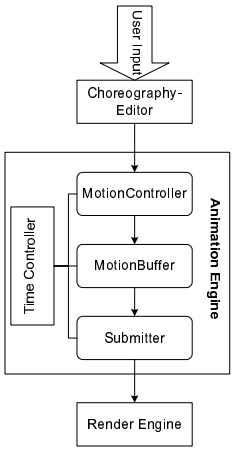

Figure 1: *The System Environment.*

The main components of the animation engine are the *MotionController*, the *MotionBuffer* and the *Submitter*. The *MotionController* holds a character model which constitutes the topology of the characters's skeleton, a set of motion models (which we explain in the next section) and a command list with the current commands generated by the choreography editor. The *MotionController* produces animation data at a fixed frame rate. i.e. at regular intervals it creates data which defines a posture of the character's skeleton. The *MotionBuffer* collects the character's animation data of the current frame and for some frames lying in the future. Finally the *Submitter* transmits the joint data of the the current frame to the *Render Engine*. The three components *MotionController*, *MotionBuffer* and *Submitter* receive the current time and are synchronized by a *TimeController*.

4. Motion Models

Motion models are abstract representations of human motion which execute atomic tasks. The idea is that every motion belongs to a certain motion model, e.g. *walk*, *run*, *wave with hands*, *throw*. Motion models are able to produce a given motion in different styles. The specific output of a

motion model is determined by a set of high level parameters. Example for high level parameters could be *happy* or *tired* which describe the mood a character has to express, or *Charley Chaplin* as a motion style in the form of a stereotype. Some parameters are shared by different motion models. E.g. every motion model has a list of the character's parts of the body which are essential for the motion and a list of those parts of the body which are not essential. For a pointing gesture the arms are important whereas the feet are not important. Other parameters are motion model specific, e.g. the direction of a pointing gesture. Every motion model has its own procedure for creating motion data driving a virtual character. These procedures are steered by parameters which are hidden in the motion model. Motion models enable the production of a variety of complex human motions without getting involved into technical details of their production. On the other hand for every motion model the transformation of the high level parameters into the specific parameters which control the production of the animation has to be implemented separately.

5. Base Motions and Clips

A motion model creates a certain motion by modifying and blending given motion data. The motion data consists of small sequences (clips) of animations which we create from motion capturing or by key frame animation. Before using the data, it is transformed into a sequence of frames corresponding to the fixed frame rate of the *TimeController*.

The way the motion data is modified and the motions are blended is controlled by parameters which are determined by a motion model's high level parameters. The motion model creates data for a virtual character which is given by a *Character Model*. The *Character Model* is defined by its joints and its skeleton. It also holds information which joints belong to a certain part of the body. The assignment of joints to a certain part of the body is crucial if the animation data of the different motion models is supposed to be blended later. Each motion model indicates the parts of the body it needs for the animation and which parts of the body are not needed. Thus there is a conflict if two motion models need the same part of the body. An example is a walk and a sit motion model. For both motion models the feet of the character are needed. At the moment such a conflict is resolved by not accepting one of the concurrent motion models.

The objects which hold the basic animation data of a motion model are called *base motions*, a term which was introduced by Grassia [4]. A *base motion* consists of the animation data (the *ClipPrimitive*) and an *Annotation* which further describes the *ClipPrimitive*. These *Annotations* contain technical data such as spatio-temporal characteristics of the motion, e.g. the time interval at which the left foot touches the ground in a walk motion. The annotation also contains information which describes the style of a motion in an abstract way. In particular we hold the information which emotional state the motions express. For developing an emotional classification we adopt research results from emotion psychology [3, 7, 8]. We use the classification by Izard [8] who proposed nine base emotions: interest, joy, surprise, sorrow, anger, disgust, contempt, fear and shame. This emotion classification has been used by Piesk and Trogemann [12] for a conversational virtual actor and by Grünvogel, Piesk and Schwichtenberg [5] as one category in an extensible classification scheme for a motion data base. Every base motion is annotated with one of these emotions.

A motion model holds all the *base motions* it needs for producing its animations. The motion model generates a specific motion by combining certain *ClipPrimitives* coming from these base motion. Consider for example the motion model *Walk* which has to produce the animation data for a walking character. In this case the motion model can contain at last three base motions. The first base motion *WalkStart* contains the part of the animation where the character starts to walk from a standing poseture, the second base motion *WalkCycle* contains a cyclic walk motion and the third base motion *WalkStop* contains the last part of the animation, which ends with a standing posture. If the motion model *Walk* now has to produce a walking character which looks happy, we produce the animation by taking three base motions *WalkStart*, *WalkCycle* and *WalkStop* having the annotation *joy*. If we want an *angry* walk we exchange the three base motions by three others having the *anger* annotation. Thus it is very easy to switch between different styles of a motion just by exchanging the base motions.

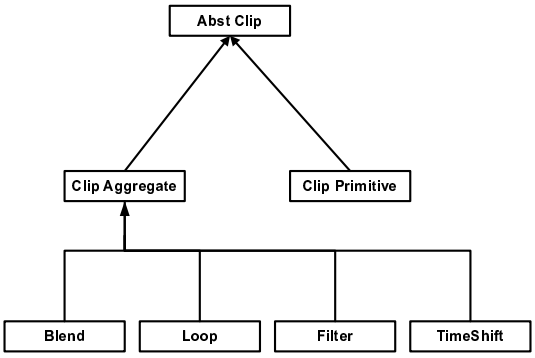

Figure 2: *Class hierarchy of the clips.*

The *ClipPrimitive* is a class which is derived from an abstract class *AbstClip* which defines the common interface to all animation clips. Every class which is derived from *AbstClip* has a method which returns the animation data for a given time. Further information is available, which indicates for which joint actually data exists, because sometimes only animation data of certain parts of the body is available. Derived from the *AbstClip* class we have the *ClipAggregate* class which serves as a wrapper. Then we derive operators on animation clips which can be applied on objects derived

form *AbstClip*. Because they are themselves derived form *AbstClip*, they can also be combined arbitrarily. The classes under *ClipAggregate* are used to blend clips or to manipulate a clip. At present we have implemented the unary operators *TimeShift*, *Loop* and *Filter* and the binary operator *Blend*. The *TimeShift* just sets the starting time of a clip to a certain frame, the *Loop* repeats the underlying animation either a finitely or an infinitely many time. The *Filter* is a more complex operator. It applies a linear filter to the translation values of the clip and a spatial orientation filter on the rotation data of the clip. Here we use the results of Lee [10] to construct a filter which works on unit quaternion signals. This *Filter* operator is used to smooth a given animation, but it can also be used to experiment with arbitrary filter coefficients to get unique effects on the animation data. To blend two base motions we have implemented a binary *Blend* operator. It can also be decided which joint's animation are blended and which are not.

6. Dynamic Motion Models

Grassia [4] built an animation system for off-line animation of virtual characters. We are interested in real-time animation with the possibility of user interaction changing the outcome of a motion model during its execution. Thus we had to adjust the concept of motion model to this stronger requirement, allowing to change dynamically the outcome of a motion model while the animation of the motion model is played.

Every motion model has its own strategy to produce its corresponding animation. We show the construction of a motion in the motion model *Walk*. First we start the motion model by specifying the number of steps the character has to walk at the beginning.

Given the starting time t_0 and the number of steps N, the motion model creates the *motion tree* as shown in Figure 3. By *motion tree* we denote the operator tree which we get if we blend and modify several clips. The leafs of the motion tree are the clip primitives *WalkStart*, *WalkCycle* and *WalkStop* indicated by squares which hold the actual animation data. In *WalkStart* the character starts to walk from a standing posture, moving only one leg. *WalkCycle* is a periodic walk cycle and *WalkStop* is the transition from the walk cycle to a standing posture. At the next layer there are *TimeShift* clips, which shift the starting time of the animations such that the *WalkStart* animation starts at time t_0 and the other two clips follow in the respective order. Above the *WalkCycle* the motion model applies a *Loop* operator, with N representing the number of repeats of the underlying *WalkCycle* clip. Next, the resulting clips of the time shifted *WalkStart* and the looped and time shifted *WalkCycle* are blended. Then this resulting clip is blended with the time-shifted *WalkStop* clip.

In an interactive environment one wants that the character starts to walk without specifying the number of steps in advance. Thus the character has to repeat the *WalkCycle* clip until a new command is invoked by the user to stop walking. As the motion model *Walk* receives the command to produce an animation starting at time t_0 it creates the motion tree as shown in Figure 4. This is a blend of the time-shifted start motion with an infinite loop of the *WalkCycle* clip. If the user wants to stop the current walk animation, the resulting motion tree of the motion model depends on the current time. If the current posture of the character was blended directly to a standing posture, this would result in a very unnatural looking motion. Thus we provide a method so that the motion model can finish its task correctly. If a stop command is received by the motion model while execution of the anima-

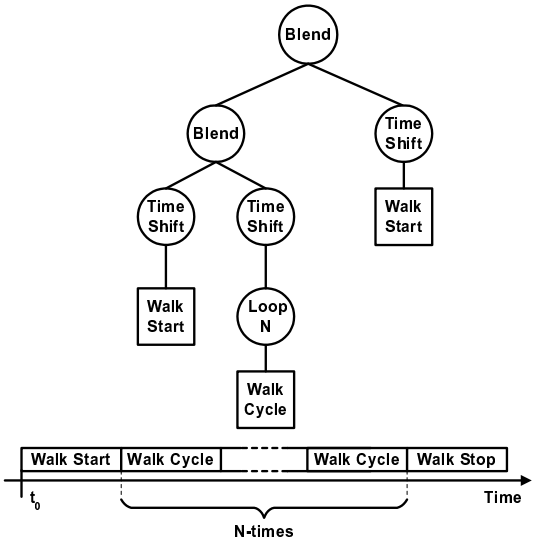

Figure 3: *Motion tree of motion model* Walk *with specifying the number of steps N.*

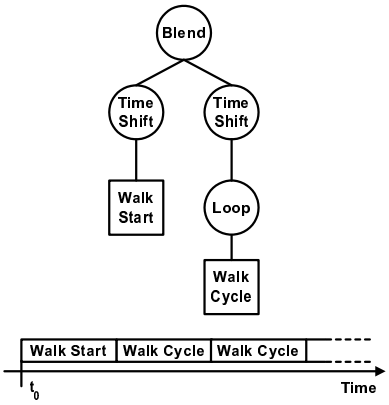

Figure 4: *Motion Tree of the motion model* walk *without specifying the number of steps.*

tion is still in the *WalkStart* clip, the motion model throws away the blend of the *WalkStart* and *WalkCycle* and returns a new tree with a blend from the *WalkStart* to the *WalkStop* clip (cf. Figure 5). Note that this is a cheap operation, if the stop command was not triggered in the blend phase between *WalkStart* and *WalkCycle*.

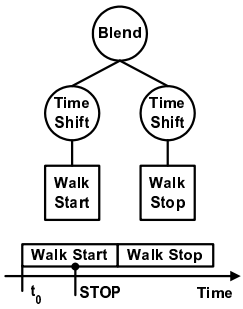

Figure 5: *Motion Tree of the motion model* walk *if stopped during the starting phase.*

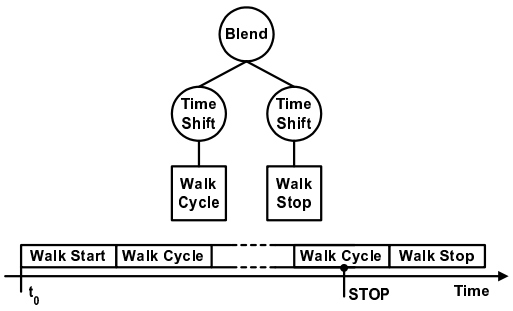

Figure 6: *Motion Tree of the motion model* walk *if stopped during the walk cycle.*

Otherwise, if we are in the *WalkCycle* phase of the motion, the first part of the motion *WalkStart* has no contribution to the motion anymore. Thus it can be skipped and we replace the infinite *Loop* clip and blend the *WalkCycle* with the *WalkStop* clip, resulting in the motion clip in Figure 6. In Figure 7 one can see the resulting animation. The character starts to walk from a standing posture and gets the stop command after it has walked some steps. Then it finishes the walk movement by excecuting the last step to end in a standing posture.

7. The MotionController

The *MotionController* receives commands which fall into two categories. The first category steers the overall behavior of the *MotionController*, e.g. resets the *MotionController* to a default state, positions the character at a default place or

Figure 7: *Result of the motion model* walk *when stopped during its execution.*

cleans the command list. The second category of commands concerns motion models which we call *MotionCommands*. *MotionCommands* concerning a motion model include the start and stop of a motion model. With *MotionCommands* a change of the motion models's parameters is possible while the motion model task is executed. The *MotionController* contains a *command list* holding the *MotionCommands* that actually influence the current animation. As the *MotionController* receives a new motion command a time stamp with the current time is added to the command and it is put into the *command list*. Before a new motion command gets inserted to the command list, the command list is updated to delete commands which are out of date, i.e. which do not have any influence on the current animation anymore. If for example there is a start command of a motion model in the *command list*, whose animation was successfully finished, the command can be deleted from the *command list*.

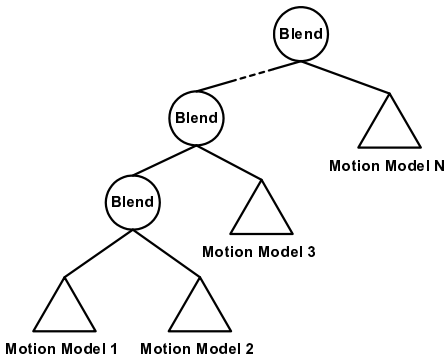

Figure 8: *Motion tree which is build from the command List. Triangles indicate subtrees which are created by motion models.*

The complete animation of a character is built from the command list. For each command the motion controller receives the motion tree from the corresponding motion model. These motion trees are blended to build the complete animation corresponding to the current command in the command list (cf. Figure 8) using the *blend* operator introduced in Section 5. Since we only have binary blend operations, the resulting motion tree is degenerated in the sense, that every right child of a knot is a motin tree resulting from a motion model. If the motion controller receives a new command and the command list gets actually changed, the current motion tree is discarded. Then the motion tree is build up completely anew from the modified command list. At the first glance this seems to be an expensive operation. But in fact only the motion models which are modified have to build a new motion tree. The other motion models have cached their current motion tree and thus only the blends between the subtrees have to be recalculated.

As an example consider Figure 7 and Figure 9. These are the animations which are produced by the motion models *walk* and *wave* executed separately. If we now start first the motion model *walk* and after a while the motion model *wave*, then a motion tree similar to the one in Figure 8 is created. The resulting animation can be considered in Figure 10.

Figure 9: *Some frames of the motion model* wave.

Figure 10: *Blend of the motion model* walk *and* wave.

At present, we just blend the motion trees of the different motion models without respecting the characteristics of different motion models. This works fine for simple motion models, but for more complex motion models one has to find special blending parameters for natural looking transitions from one movement to another. Here further research has to be done, and it is thought of a mechanism for finding automatically the right blending procedure for different motion models.

8. Experimental Results

We have implemented an experimental version of the animation engine with C++ under Linux and under Windows NT 4.0 running on a PC with an 1100 Mhz AMD processor. The graphical output of the *Render Engine* is conducted with OpenGL. For test purposes we have used a rather complex character with 67 joints and a detailed geometry with about 9000 Polygons and 3.6Mb texture. At present, we have implemented the motion models *stand*, *walk*, *squat*, *wave* and *sidestep* and use 14 *base motions*. The *base motions* are generated by keyframe animation and contain animation data for the character which are designed for an animation rate with 30 frames per second. The data of the *base motions* is given in biovision file format, which is read in by the animation engine and converted to the appropriate *base motion* objects. We use the computer's keyboard to send *MotionCommands* to the animation engine.

As a result, the performance of the system is promising. We have not found delays which could come from the building or the reorganisation of the motion trees, if new commands are sent to the motion controller. It seems, that at the moment rather the graphical output limits the performance of the system which is caused by the complex geometry.

9. Conclusions and further research

Motion models provide a practical abstraction for the creation of animated virtual characters. Their interface hides the complex construction of the animation and thus facilites the production of the same animation in different styles and for different purposes. The motion models can create motions in different styles by changing their base motions. The *clips* classes as presented in Section 5 serve as simple building blocks for the construction of the motion models. The *ClipPrimitive* which actually hold the animaiton data only have to be exchanged to create different styles of the motion. They *clips* also are used for the combination of the animation of different motion models. We have shown, that dynamic motion models can be applied for real-time animation. A simple approach for the creation and modification of motion trees is proposed, producing complex motions for virtual characters.

The drawback of motion models is, that it can be a rather demanding task to implement new ones if the modeled motion has many constraints.

Further research has to be done on how to create naturally looking transitions between different motion models automatically. We also have to adopt a mechanism allowing the real-time fulfillment of constraints to the character and implement and improve the clip operators. Furthermore a generic motion model framework would facilitate the production of new motion models.

Acknowledgments

This work was supported by the BMBF grant 01 IR A04 C (M^3 -Eine mobile Multi-User Mixed Reality Umgebung) and the BMBF grant 01 IL 904 L (EMBASSI - Elektronische multimediale Bedien- und Service-Assistence).

References

1. Norman I. Badler, Cary B. Phillips, and Bonnie Lynn Webber, *Simulating humans: Computer graphics and control*, Oxford University Press, 1993. 1
2. Armin Bruderlin and Lance Williams, *Motion signal processing*, Computer Graphics **29** (1995), 97–104. 2
3. P. Ekman, W. V. Friesen, and P. Ellsworth, *What emotion categories or dimensions can observers judge from facial behavior?*, Emotion in the Human Face (P. Ekman, ed.), Cambridge: Cambridge University Press, 2 ed., 1982, pp. 39 – 55. 3
4. F. Sebastian Grassia, *Believable automatically synthesized motion by knowledge-enhanced motion transformation*, Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, 2000. 1, 2, 3, 4
5. Stefan Grünvogel, Jens Piesk, and Stephan Schwichtenberg, *AMOBA: A database system for annotation captured human movements*, Computer Animation 2002, IEEE Computer Society Press, 2002. 3
6. Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien, *Animating human athletics*, Computer Graphics **29** (1995), 71–78. 1
7. C. E. Izard, *Human emotions*, Plenum, 1977. 3
8. _____, *The psychology of emotions*, Plenum Press, New York, 1991. 3
9. Prem Kalra, Nadia Magnenat-Thalmann, Laurent Moccozet, Gael Sannier, Amaury Aubel, and Daniel Thalmann, *Real-time animation of realistic virtual humans*, IEEE Computer Graphics and Applications **18** (1998), no. 5, 42–57. 2
10. Jehee Lee, *A hierarchical approach to motion analysis and synthesis for articulated figures*, Ph.D. thesis, Korea Advanced Institute of Science and Technology, Department of Computer Science, 2000. 2, 4
11. Ken Perlin and Athomas Goldberg, *Improv: A system for scripting interactive actors in virtual worlds*, Computer Graphics **30** (1996), 205–218. 2
12. Jens Piesk and Georg Trogemann, *Animated interactive fiction: Storytelling by a conversational virtual actor.*, Proceedings of VSMM'97, IEEE Computer Society Press, 1997, pp. 100 – 108. 3
13. Gael Sannier, Selim Balcisoy, Nadia Magnenat-Thalmann, and Daniel Thalmann, *"VHD: A system for directing real-time virtual actors*, The Visual Computer **15** (1999), no. 7/8, 320–329. 2
14. Ken Shoemake, *Animating rotation with quaternion curves*, Computer Graphics (Proc. of SIGGRAPH '85) **19** (1985), no. 3, 245–254. 2
15. Xiaoyuan Tu, *Artificial animals for computer animation*, ACM Distinguished Theses, Springer Verlag, 1998. 1
16. Xiaoyuan Tu and Demetri Terzopoulos, *Artificial fishes: Physics, locomotion, perception, behavior*, Computer Graphics **28** (1994), 43–50. 1
17. M. Unuma and R. Takeuchi, *Generation of human motion with emotion*, Computer Animation '93, Proceedings, 1993, pp. 77–88. 2
18. Andrew Witkin and Zoran Popović, *Motion warping*, Computer Graphics **29** (1995), 105–108. 2