CS585
Term-Paper

Dynamic Simulation Methods for Animation of Legged Locomotion

Daniel Nikovski

Abstract

The modeling of legged locomotion is of central importance to the successful animation of human and animal figures. Since legged locomotion is typically difficult to animate using only kinematic modeling, alternative, more physically-based methods have to be employed. This paper discusses several methods for dynamic simulation of legged locomotion. Typical components of a dynamic simulation system are outlined, and a number of successful applications are reviewed.

1 Introduction

All methods for the animation of legged locomotion can be split into two broad categories: those based on the laws of motion without considering the actual forces that cause the motion (kinematic modeling methods) and those taking into account the physical nature of the motion (dynamic modeling methods). This distinction is due partly to a long tradition in mechanics and partly to the difference in computational techniques used to implement the two types of methods.

A further distinction within these two classes is made between forward and inverse modeling. A legged figure is represented as a number of bodies connected together by means of joints. The position and orientation of the figure are specified by parametrs of the joints (for rotational joints, the parameters are rotation angles; for prismatic joints - offsets). A single position and orientation of the body parts exists for each configuration of the parameters. The problem of finding this position and orientation is referred to as the *forward kinemtic problem*. The inverse problem of finding a parameter configuration that results in a given position and orientation is accordingly called *inverse kinematic problem*.

While the forward kinematic problem is usually straightforward, the inverse kinematic problem is notoriously hard to solve. This is due to the fact that the inverse kinematic problem is often ill-posed, i.e., its solution is not unique - instead, a whole set of possible parameter configurations exist that satisfy the given position in space. Analytical solution is possible only in a small number of special cases. Otherwise, numerical methods have to be employed, which often have erratic behavior due to the ill-posedness of the problem [27].

Similarly, forward and inverse dynamic problems can be defined. Forward dynamic problem is the task of finding position and orientation in space as well as the velocities of the body parts that correspond to given forces and torques applied to those parts. Inverse dynamics is the process of determining what forces and torques would produce the desired position and velocity of the body parts. A difference between the inverse kinematic problem and the inverse dynamic problem is that the latter does not deal with Cartesian space coordinates - instead, a solution for the forces and torques that lead to a given parameter configuration is sought only. Thus, the inverse dynamic problem is always preceded by solving the inverse kinematic problem, and consequently inverse dynamic modeling possesses all the drawbacks of inverse kinematic modeling.

It should be noted that no significant difference exists between the order of computations required to solve the forward and inverse dynamic problems. For both types of problems, algorithms have been proposed that are linear in the number of joints [17, 18].

Conditionally, the four modeling methods discussed above can be ranked in increasing order of computational complexity as follows:

1. Forward kinematic modeling.
2. Forward dynamic modeling.
3. Inverse kinematic modeling.
4. Inverse dynamic modeling.

2 Dymamic vs. Kinematic Modeling for Legged Locomotion

The order above suggests that whenever possible, forward kinematic modeling should be tried first. What makes legged locomotion special in this regard is the fact that generally it cannot be implemented through forward kinematic modeling because of a fundamental constraint - the legs have to touch the floor and never penetrate it, if the animation has to be realistic. This constraint is very hard not to violate if only joint angles have to be specified. The animator cannot easily impose this constraint when working in the domain of parameters, because this constraint is no longer simple in that space [27].

If kinematic modeling only is to be used, the modeling should be inverse. This leads to the aforementioned computational difficulties, and hence kinematic modeling is not a good choice for legged locomotion. On the other hand, it is relatively easy to implement the floor constraint through forward dynamic modeling. This is the rationale behind using dynamic instead of kinematic modeling for the animation of legged locomotion.

3 Structure of a Physically-Based Animation System

A dynamic simulation method has the following three main components:

Physical model. Usually the physical model is described as a set of ordinary differential equations (ODEs) that reflect the dynamic interactions between the masses in the system. The two major physical model alternatives are particle-based systems [6] and rigid body systems [9].

ODE solver. A variety of methods can be employed to solve the ODE system [16, 7]. The output of the ODE solver is a sequence of positions of the body parts that can be fed further into the viewing and rendering pipelines.

Control system. Unlike kinematic modeling, interesting behavior can be obtained through dynamic modeling without any dedicated control system. For example, a ball bounces in a physically realistic way without being explicitly controlled. However, in order to achieve more interesting motion of the animated objects, the forces acting upon them have to be modified in the course of the animation by a control system.

Each of these components is discussed shortly below.

4 Physical Models for Animation

The dynamic sumulation of jointed figures is based on the laws of mechanics established by Isaac Newton. If the animated objects are treated as material points, the resulting equations of motion are relatively simple. Such simulation is called particle-based.

If the geometry and mass distribution of objects are taken into consideration, the equations of motion are substantially more complex. This approach is referred to as rigid body simulation.

4.1 Particle Systems

The relationship between the acceleration $\mathbf{a}$ of a body with mass m due to a force $\mathbf{f}$ is given by

$$\mathbf{a} = \frac{\mathbf{f}}{m}$$

Having in mind that the acceleration $\mathbf{a}$ is the second derivative of the position of the body $\mathbf{x}$, the equation can be rewritten as

$$\ddot{\mathbf{x}} = \frac{\mathbf{f}}{m}$$

This can be reduced to Cauchy normal form by the substitution $\dot{\mathbf{x}} = \mathbf{v}$. The result is a set of six ODEs:

$$\dot{\mathbf{v}} = \frac{\mathbf{f}}{m}$$
$$\dot{\mathbf{x}} = \mathbf{v}$$

If we assume that the mass of the whole system is concentrated in a small number of material points, the equations above can be applied immediately. Their solution will determine the state of the system at any moment in time.

The forces that act upon these mass points can come from a variety of sources. The most basic force is the force of gravity that acts downwards proportionally to the mass of the particle. In order to implement nontrivial behavior, however, this force only is not enough.

It was discovered that a simple mechanism exists for implementing interesting forces in a particle system - connecting the particles with springs. A spring has a resting length r and resists any stretching or contracting by elastic force according to Hook's law [6]:

$$\mathbf{f} = -\left[k_s(|\mathbf{l}| - r) + k_d\frac{\dot{\mathbf{l}}\cdot\mathbf{l}}{|\mathbf{l}|}\right]\frac{\mathbf{l}}{|\mathbf{l}|}$$

This elastic force acts upon the two particles that are connected by the spring. Here $\mathbf{l}$ denotes the current length of the spring, k_s is the spring constant, and k_d is a damping coefficient. Other, more general spatial forces can be implemented that act upon more than two particles [6].

If legged locomotion is implemented through particle systems, a number of additional constraints have to be realized. One way to achieve constraints like keeping particles a specified distance apart is to connect them through very stiff springs. This approach has the drawback that the resulting ODEs are hard to solve and generally unstable.

An alternative approach is to use *penalty functions* that become zero if and only if the constraint is satisfied. An energy function is formed, whose gradient is an abstract force that tends to keep the constraint satisfied. [5]. The drawback of this method is that the constraints are not strictly kept; rather than that a balance between the different constraints is maintained, so that the total energy function is minimized.

Still another alternative is to use *constraint forces* that will counteract the existing forces in such a way that the constraints will be kept precisely. It is possible to generate these forces automatically from constraint specifications [8].

4.2 Rigid Body Systems

If the simulated objects are considered with their spatial extents, the material point idealization employed in particle-based systems is no longer appropriate. Newton's law cannot account for the changes in orientation that the objects undergo and it has to be supplemented by laws that govern angular velocity and acceleration. Such a law is Euler's law that gives the moment N that must be acting on a body to cause angular velocity ω and angular acceleartion $\dot{\omega}$:

$$N = {}^C I\dot{\omega} + \omega \times {}^C I\omega$$

Here ${}^C I$ is the inertia tensor of the body computed with respect to its center of mass. The inertia tensor describes compactly the mass distribution of the body [17].

Using Newton's and Euler's equations, an iterative solution of the forward and inverse dynamic problems can be obtained [17]. The Newton-Euler dynamic formulation can be used to derive closed-form solution for both problems as well.

An alternative way to describe the dynamics of a set of rigid bodies is to use the Lagrangian dynamics formulation. It is based on energy conservation principles and derives the same equations of motion that the Newton-Euler folmulation would, albeit in a different way. The computational efficiency of the two formulations, however, can be very different - a difference in orders of magnitude is possible [17].

Efficiency concerns gave rise to many enhanced algorithms that use special computational arrangements to cut on the number of multiplications and additions needed to model the dynamics of the object. Among the most successful known algorithms for forward dynamic simulation are the Composite-Rigid-Body Method and the Articulated-Body Method that are based on a mathematic abstraction called Spatial Vector Analysis [18].

5 Integration of the Equations of Motion

The purpose of the ODE solver is to integrate the equations of motion obtained through the methods described in the previous section. The result of this integration is a time sequence of positions in space of the objects being animated. Each element of this sequence is a "snapshot" of the animation scene for a particular moment in time - accordingly, these snapshots can be rendered further to produce animation frames.

Several requirements are put forth to the ODE solver. It should be computationally efficient, since it will be invoked between each pair of successive frames. Furthermore, it should be stable, accurate and consistent with the physical problem being solved. Many methods for the solution of ODE have been proposed [16]. Each of them is a compromise between the four requirements mentioned above.

The simplest integration scheme is the forward Euler method [7]. It is very efficient computationally, but is susceptible to stability and accuracy problems. This is due to the low order of the method and the explicit integration employed. The method is straightforward to implement, because it requires only one evaluation of the right-hand side of the system of ODEs at its current state.

Interestingly, this integration scheme enjoys popularity in dynamics-based animation graphics inspite of its apparent drawbacks. One possible explanation is that in animation accuracy and consistency are not of primary importance - small deviations from the true solutions are permissible, so long as the qualitative nature of the motion is preserved. On the other hand, computational efficiency is always very important, because a vast number of integrations are performed. This biases the choice towards this method.

The only real problem with the forward Euler method for the purposes of animation is its instability. Physical processes that in reality are convergent to a single point or a limit cycle in state space can actually diverge if integrated by the forward Euler method. This might change the qualitative nature of motion and make the animation unrealistic. For example, a falling ball might bounce higher and higher in successive iterations, even when friction is taken into account and the physical process is supposed to be damped.

The Runge-Kutta integration scheme provides a solution to the stability problem. It is relatively more computationally expensive than Euler integration if the same step size is employed. However, this integration scheme permits much larger steps to be taken, which eventually leads to savings in computations. The fourth-order Runge-Kutta method has been known as the best trade-off between accuracy and efficeincy for most problems in science and engineering [16, 25].

Other integration schemes can be used as well. Bulirsch-Stoer integration is very efficient for smooth functions - it can increase adaptively its step size to take advantage of flat regions and reduce it to explore carefully more complex ones. This feature cannot be employed fully in animation, though, because there is an upper limit for the step size - the time between two frames. Taking steps larger than that will require inbetweening, which might be even more computationally expensive than integration. Besides, the equations of motion are typically not smooth - for example, when a collision between two bodies occurs, the right-hand side of the equations changes abruptly.

For cases when the equations of motion are particularly discontinuous, methods for the solution of stiff ODEs have to be employed [16, 25]. Arguably, these methods are particularly relevant to dynamic animation, because the system of ODEs that correspond to an animated scene are stiff if, for example, different objects in the scene have very different speeds of motion. It should be noted that these methods are not necessarily computationally more expensive - on the contrary, savings of orders of magnitude can be realized in comparison to usual methods when stiff problems are solved [16].

6 Control of Legged Locomotion

In order to achieve legged locomotion in dynamic systems, some kind of active forces should act upon the animated objects. These forces can be produced by motors, muscles or some other type of actuators that convert potential energy into kinetic one. Systems that possess such actuators are called *articulated systems* to distinguish them from passive systems that are acted upon by forces, but cannot power and regulate their motions [2].

Many methods exist for the control of articulated dynamical systems. These methods can be split generally into adaptive and non-adaptive ones. The major distinction between the two classes is whether the designer has to specify the control algorithm in detail or the controller can "learn" to regulate the system by itself. Recent advances in learning and adaptive computation theory have made possible the application of algorithms of the latter type to a growing class of control problems, including the animation of legged computation.

6.1 Classical and Modern Control

Most often dynamical systems are controlled by control laws designed according to the methods of classical and modern control. These methods are applicable if a sufficiently precise model of the dynamic system being controlled is available. Any inaccuracies in the model impair the performance of the controller, so precise models are a major requirement for the applications of these methods. Accordingly a lot of human effort has to be expended in order to obtain a good controller. Some control problems might be so difficult to analyse and provide a model for that the application of classical and modern control techniques might be infeasible.

6.2 Adaptive Control

An alternative approach is to use controllers that adapt their structure and control laws so that their performance increases with time. These adaptive controllers can be simply parametrized versions of classical controllers that adjust their parameters in order to achieve better control. Other possibilities include the use of neural networks, fuzzy logic and evolutionary computing.

The last three approaches treat the controller as an universal function approximator (UFA). The function being approximated is the law that maps the state variables of the system to control influences - that is, the control law of the system.

6.3 Neural Networks

Neural networks have been used in many control problems as UFAs. Supervised learning is employed when the correct output for a given state of the system is known and can be given as an example to the network [19]. This is rarely the case, actually, because a known correct output presumes that a correct controller already exists to provide it. Clearly this technique is not very useful in control.

The supervised learning approach, however, can be augmented with stochastic behavior on part of the network, which will let it learn the correct output even when only a single scalar variable is available to the network instead of a whole output vector, so long as this variable reflects how appropriate the output of the network was. This variable is called *reinforcement* and the method correspondingly *reinforcement learning*.

This model is very suitable for learning control problems, because the reinforcement variable can be directly measured from the performance of the controller without having to identify what the actual correct output is. By manipulating the reinforcemant variable the controller is either "rewarded" or "punished" for its actions until it learns the control law. Associative Reward-Penalty [26] and the REINFORCE class of reinforcement learning algorithms [23] are examples of methods that employ this learning scheme.

A complication arises when the environment delays the reinforcement. For example, if a robot balances a broomstick and makes a wrong hand movement, the negative reinforcement will come only after the broomstick has fallen down, before which the robot will have made many more movements. It is very difficult to determine exactly which of these hand movements was wrong and led to the loss of balance, so that it can be punished. In this case we have the problem of *temporal* credit assignment in addition to the usual *structural* credit assignment problem. Structural credit assignment deals with determining the change in which connection led to better performance, while temporal credit assignment deals with determining *when* (at which moment in time) correct outputs have been generated and when the outputs have been incorrect.

Examples of reinforcement learning algorithms that solve the temporal credit assignment problem are TD(λ) [21] and Q-learning [22].

6.4 Fuzzy Logic

Fuzzy logic is another important tool for building controllers [20]. The operation of fuzzy logic control systems can be explained in many ways, but I believe it is best understood if fuzzy logic systems are viewed as an extension to the rule-based architectures of classical AI.

Rule-based systems employ a set of rules that prescribe how the system should react to a particular input. When the rule will fire is determined by the *antecedent* of the rule which is usually a logical expression that is true only for some values of the input. If rule-based systems are to be used to the ends of control, there should be a rule for every possible value of the input to the system. On the other hand, if precise control has to be exercised, each rule should fire for a small set of inputs. Together these two requirements result in a huge number of rules for any control problem of nontrivial complexity.

Another problem is the discontinuity of the control law that results from the application of a rule-based approach. Special care should be taken to guarantee that the control law does not change abruptly at the boundary between the action regions of two rules.

Fuzzy logic provides an elegant remedy to these two problems. Fuzzy rules differ from traditional expert systems rules in that the antecedent matches the state of the system with a certainty value between 0 and 1. This certainty value serves as a weight factor for the influence of the consequent part of the respective rule to the final control response of the system. Furthermore, not only the best-matching rule is taken into considerations, but all the rules in the system have their influence upon the output. In this way the output of the system "blends" smoothly the consequents of the different rules when the input of the system changes.

The major drawback to fuzzy logic systems is that they cannot self-tune their performance, i.e., they are not truly adaptive. Even though it has been proven that fuzzy systems are UFAs, no powerful methods have been proposed to train them except schemes that work for a limited number of problems. Most successful industrial applications use either handcrafted rules or rules generated by a simple algorithm and then adjusted manually.

An interesting idea is to use a learning method based on neural newtorks, evolutionary computation (see below) or some statistical scheme to generate the rules. An advantage to this approach is that the generated rules can be analyzed and accepted, rejected or enhanced, as well as mixed with hand-crafted rules. This feature together with the possibility to implement the rules in specialized hardware makes fuzzy logic systems an efficient and powerful tool for control.

6.5 Evolutionary Computation

The name Evolutionary Computation refers to a group of related techniques based on the principles of natural selection that can perform search and optimization [24]. Since adjusting the parameters of an adaptive controller is basically optimiztion in the space of these parameters, these techniques are particularly relevant to the problem of adaptive control.

Essential to this approach is the use of a *fitness* function that guides the process of evolution. Its purpose is much similar to the purpose of the reinforcement variable in reinforcement learning - to promote good solutions and designs and supress bad ones. The fitness function is used to determine which solutions in a pool will "survive" and continue development. Different operators are used to produce new individuals from successful old ones. The process is repeated until satisfactory performance is achieved.

Several main styles of EC are known, distinguished mainly by the types of structures that form the population [14] - Genetic Algorithms, Genetic Programming, Evolution Strategy, Evolution Programming. Genetic Algorithms operate on *genotypes* that are simply an encoded description of the organism or *phenotype*. The individuals in Genetic Programming are not strings of bits, but complete programs in the form of trees (usually LISP programs). In Evolution Strategy the individuals are evaluated and evolved directly, without the help of genotypes; the description is in the form of real values. Evolutionary Programming extends the idea of Evolutionary Strategy by operating on a larger variety of representations, including finite-state machines.

These approaches differ also in the evolutionary operators that are applied to the structures in the pool in order to obtain new structures. All of them use *mutation*, or random change in the values of the parameters being evolved. With the exception of Evolutionary Programming, all other three styles use *crossover*, or recombination of genetic material from two "parents". In addition to that, Genetic Programming uses variations of recombination other than crossover.

A characteristic feature of EC is the ability to arrive at global rather than local maxima of the fitness function, a feature that traditional optimization schemes do not possess. This makes EC a very powerful optimization tool.

A very promising idea also is to combine evolutionary algorithms with reinforcement learning . In practice, a neural net is produced, after which it is trained, then the fitness is evaluated and the net evolved. The new element here is that the neural net is not expected to produce the proper control behavior right away - instead it is trained first, which highly increases its chances to receive good fitness value thereafter. The relationship between neural net training and genetic algorithms is the same as the one between life-long learning and evolutionary modification in living organisms.

7 Applications

The techniques for dynamic simulation and control in the previous two sections have been used in several animations of legged locomotion. This section reviews some of them.

7.1 Girard and Maciejewski

The PODA animation system, developed by the authors, employed both kinematic and dynamic modeling to achieve smooth human figure locomotion [11]. Their primary concern was to make the motion look as if forces were being applied and not fully simulate these forces.

The control algorithms that they used were based on the decomposition schemes proposed by Raibert [11]. The decomposition in the PODA model is performed along two lines: decomposition by leg and decomposition by body direction.

Girard and Maciejewski discovered a simple way to animate multi-legged figures that produced very realistic results. They superimposed the ballistic motion of the body due to each leg independently to arrive at a single trajectory. This approach gave good results even in cases when the motion of the individual legs was different - for example, if one of the legs pushed extra hard, the net effect was of the body being propelled into the air, thus producing running gait.

In PODA the horizontal direction path is specified by the animator with a cubic spline (Catmull-Rom or B-spline). The accelerations are adjusted according to the desired speed at the individual frames.

This approach used articulated models, classical control algorithms and numerical integration techniques.

7.2 Beer

Randalf Beer created a dynamic environment for simulation of behavior of artificial creatures. His primary goal was to investigate the processing that takes place in a simulated nervous system, but he maintained that nontrivial processing will emerge only in the course of interaction of the simulated agent with the environment.

His simulation employed simplified laws of physics, in which velocity and not acceleration was proportional to force. He designed a cockroach that was able to walk with different gaits and form comlex trajectories. The control was effected by an artificial nervous system that was carefully designed to produce the desired gaits. In later experiments the nervous system was subjected to evolution and its performance improved.

Even though the animation was only 2D and the system used simpified laws of physics, the work of Beer is important because of the approach taken to build the control system. It seems that the combination of neural networks and genetic algorithms can provide automatically control systems of arbitrary complexity (which comes as no surprise because this is the approach taken by nature).

7.3 McKenna and Zeltzer

McKenna and Zeltzer simulated dynamically the locomotion of a hexapod using an articulated model and biologically-based control mechanisms [1]. The Articulated Body Method was used to model the hexapod as a branching structure without closing loops. The resulting ODE system was integrated with a modified fifth-order Runge-Kutta method with adaptive step size, and the positions of the body parts rendered appropriately.

A key factor to the success of their simulation was the development of a control mechanism that was able to regulate and coordinate the motion of the body parts. The control mechanism had two components - a gait controller and a set of motor programs. The gait controller was responsible for the generation of commands that would result in coordinated motion with different gaits. The motor programs executed these commands by producing the necessary control forces.

The gait controller was designed systematically, following research hypotheses and data from entomology. The change from stepping phase to stance was regulated by coupled oscillators. Synchrony was obtained by mutual inhibition between pairs of legs that were supposed to be in different phases.

The design of the motor programs, however, was not so systematic. The motor programs used exponential strings to create forces, by changing their resting lengths. The exponential strings were described by a set of control parameters, whose correct values were determined by trial and error. Certainly, such an approach requires considerable human effort and is not very practical.

McKenna and Zeltzer noted that in the process of tuning the control parameters the human expert played the role of the natural selection in evolution. They suggested that simulated evolution can be applied to the problem, but did not report any results in this regard.

7.4 Raibert and Hodgins

Raibert and Hodgkins managed to animate dynamically a number of complex gaits, including such that did not maintain static balance [2]. All motions were generated by formulating physical models, deriving equations of motion and integrating them numerically. The equations of motion were generated with a commercially available program and integrated using Euler's method with very small time step - 0.0004 s.

The control of the objects was performed with algorithms used to control real robots. This saves time designing the algorithms anew, but nevertheless the transfer is not automatic. A number of parametrs had to be "tweaked" in order to obtain the desired behavior. This is a major drawback to the approach, because highly skilled control engineers would have to be employed for successful animation.

However, Raibert and Hodgins identified certain regularities and a set of basic elements in the process of controller design. These common elements include a symmetry principle used for balance, decomposition of the control algorithms into separate parts for regulating hopping, speed and posture, and the use of elastic storage in the legs. The authors suggested that these common elements might serve as a toolbox for designing new algorithms with "reasonable" effort. How much effort is reasonable is not very clear, though; for an animator probably *any* effort in this respect is unreasonable.

Raibert and Hodgins pointed out that these difficulties might be overcome by optimization techniques and modern control theory. Several directions seem promising: penalty functions and task constraints, which were discussed in the section on physical modeling.

7.5 Sims

Karl Sims used genetic algorithms and dynamic simulation to evolve different types of legged locomoton - walking, jumping, etc. [13]. His creatures were also result of simulated evolution and their shape was not anthropomorphic; rather, they tended to take the shape that was most suitable to the artificial world that they inhabited. Their locomotion can be only conditionally called "legged".

The basic idea underlying Sims' work was the use of evolution as a generator of complexity. In this way virtual creatures can be created without understanding on the part of the animator of the processes that underlie their motion. This creates a potential problem - if the characteristics of the desired motion cannot be captured well in the fitness function employed in the evolution, the desired behavior will not evolve. For example, requirements for "smooth" and "realistic" motion are inherently hard to embed in a fitness function. We can argue that certain general types of behavior evolved as a result of social interaction - for example, different manifestations of body language. The simulation of these social interactions so that they can be used in an artificial evolution seems an infeasible task, at least presently.

All of the creatures in Sims' experiments were articulated bodies whose dynamics were computed using the aforementioned Articulated Body Method. The numerical integration was performed through an adaptive step Runge-Kutta-Fehlberg method. The simulation employed a collision response scheme that used both impulses and penalty spring forces.

8 Conclusion

The applications discussed above demonstrate that legged locomotion can be realistically animated by employing physically-based simulation methods. The dynamics of motion can be simulated automatically, which permits large-scale animations to be created.

Controlling legged locomotion, however, proves to be not as amenable to automatic design as motion simulation. Presently, considerable effort is required to build locomotion controllers that can regulate the movement in a realistic manner. It seems that in order to achieve good animation of legged locomotion, the animated legged objects have to possess a fair amount of expertise about their environment, or, in other words, be full-fledged autonomous agents that can navigate in their worlds. The research in simulation of adaptive behavior might provide useful methods in this line.

References

[1] McKenna, Michael and David Zeltzer (1990). Dynamic Simulation of Autonomous Legged Locomotion. *Computer Graphics*, 24(4):29-38, August 1990.

[2] Raibert, M.H. and Jessica K. Hodgkins (1991). Animation of Dynamic Legged Locomotion. *Computer Graphics*, vol. 25, No. 4:349-358.

[3] Raibert, M.H. (1985). *Legged Robots that Balance*. Cambridge: MIT Press.

[4] Hahn, James K. (1988). Realistic Animation of Rigid Bodies. *Computer Graphics*, vol. 22, No. 3:299-308.

[5] Witkin, Andrew, Kurt Fleischer, Alan Barr (1987). Energy Constraints On Parametrized Models,*Computer Graphics*, vol.21, No.4, 1987.

[6] Witkin, Andrew (1991). Particle Systems Dynamics, *SIGGRAPH'91 Tutorial on Particle-Based Systems*, Section C. ACM Press.

[7] Witkin, Andrew (1991). Differential Equations Basics, *SIGGRAPH'91 Tutorial on Particle-Based Systems*, Section B. ACM Press.

[8] Witkin, Andrew (1991). Constrained Dynamics, *SIGGRAPH'91 Tutorial on Particle-Based Systems*, Section G. ACM Press.

[9] Baraff, David (1991). Rigid Body Concepts, *SIGGRAPH'91 Tutorial on Particle-Based Systems*, Section H. ACM Press.

[10] Bruderlin, A. and T. W. Calverts (1989). Goal-Directed, Dynamic Animation of Human Walking. *Computer Graphics*, vol. 23, No. 3:233-242.

[11] Girard, M. and A. A. Maciejewski (1985). Computational Modeling for the Computer Animation of Legged Figures. *Computer Graphics*, vol. 19, No. 3:263-270.

[12] Armstrong, W. W., M. Green and R. Lake (1987). Near-Real-Time Control of Human Figure Models. *IEEE Computer Graphics and Applications*, vol 7, No. 6:52-61.

[13] Sims, Karl (1994). Evolving Virtual Creatures,*Computer Graphics* No.8, 1994, pp.15-22.

[14] Kinnear, K.E. (ed.) (1994). *Advances in Genetic Programming*, MIT Press.

[15] Beer, R.D. (1990). *Intelligence as Adaptive Behavior.* Boston: Academic Press.

[16] Press, W.H et al. (1992) *Numerical Recipes in C*, Cambridge University Press.

[17] Craig, John J. (1989) *Introduction to Robotics*, Reading: Addison-Wesley.

[18] Featherstone, Roy (1987) *Robot Dynamics Algorithms*, Boston: Kluwer Academic Publishers.

[19] Widrow, B. and M.E. Hoff (1960). Adaptive Switching Circuits. In *1960 IRE WESCON Convention Record*, part 4, 96-104. New York: IRE.

[20] Kosko, B. (1992). *Neural Networks and Fuzzy Systems: a Dynamical Approach to Machine Intelligence.* Englewood Cliffs, NJ: Prentice Hall.

[21] Sutton, R.S. (1988). Learning to Predict by Methods of Temporal Difference. *Machine Learning 3*, 9-44.

[22] Watkins, C.J.C.H., and Dayan, P. (1992). Q-Learning. *Machine Learning 8*, 279-292.

[23] Williams, R.J. (1992). Simple Statistical Gradient- Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning 8*, 5-31.

[24] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimiztion, and Machine Learning.* Reading: Addison-Wesley.

[25] Hoffman, Joe D. (1992). *Numerical Methods for Engineers and Scientists*, New York: McGraw-Hill.

[26] Barto, A.G. and P. Anandan (1985). Pattern Recognizing Stochastic Learning Automata. *IEEE Transactions on Systems, Man and Cybernetics 15*, 360-375.

[27] Watt, Alan and Mark Watt (1992). *Advanced Animation and Rendering Techniques*, New York: ACM Press.