Automated Learning of Muscle Based Locomotion Through Control Abstraction

Radek Grzeszczuk

Department of Computer Science
University of Toronto
Toronto, Ontario, Canada

A Thesis submitted in conformity with the requirements
for the Degree of Master of Science in the
University of Toronto

Abstract

We present a new technique that automatically synthesizes realistic locomotion for the animation of physically-based models of living creatures. The method is suitable for highly deformable creatures such as snakes and fish with many degrees of freedom and a considerable number of internal muscle actuators. The technique is a multilevel learning procedure which first performs a global search for time-sampled actuator activation functions that produce efficient locomotion. The process then abstracts these low-level activation functions into a highly compact representation. The representation emphasizes the natural periodicities of the derived muscle actions and makes explicit the coordination among multiple muscles that has led to effective locomotion. Finally, the synthetic creatures can put into practice the compact, efficient controllers that they have learned. The learning technique enables them to accomplish higher-level tasks specified by the animator and guided by simple sensory perception—for example, locomote and execute turning maneuvers to reach a visible target.

Acknowledgments

Acknowledgements to my supervisor, Demetri Terzopoulos, for his patience, ideas and support and to Eugene Fiume for introducing me to computer graphics and for being the second reader. Many thanks to David Mandel and his parents for being so helpful and for treating me as part of the family. Thank you to all the people from the graphics lab and everybody else who in some way contributed to this thesis.

Contents

List of Figures

Chapter 1

Introduction

The animation of animals in motion is an alluring but difficult problem. With the advent of visually realistic models of humans and other creatures, even small imperfections in the locomotion patterns can become distracting nuisances. The best way to achieve a satisfactory level of authenticity in the long run is to build physically realistic models which can closely approximate the movements of natural animate bodies through the use of internal actuators. Using the laws of physics to generate motion greatly increases realism but it tends to hamper our ability to control the course of the action. Sophisticated animate models must assume more of the responsibility for their own control—they must learn ways to control themselves. The sensory based motor control problem becomes increasingly challenging as the complexity of the animated creature grows. At a certain level of modeling sophistication, finding good solutions from scratch is so costly that it becomes essential not only to automate the search for solutions, but also to abstract good solutions for subsequent reuse.

The problem that we address in the thesis may be stated simply: Given a sufficiently rich physically-based model of an animate creature with internal muscle actuators capable of producing realistic locomotion, is it possible for the model to learn from first principles how to control its actuators in order to locomote in a natural fashion? In contrast to most of the prior work on variations of this problem, we are targeting animate models of at least the level of complexity and realism of the snakes and worms of Miller [23] or the fish of Tu *et al.* [37]. In both of these cases, the controllers responsible for the locomotion have been carefully hand crafted using knowledge gleaned from the biomechanics literature [21] and long hours of experimentation. We would like to devise algorithms that can give such models the ability to learn automatically how to locomote.

At the foundation of our approach lies the notion that natural motion patterns are energetically efficient. This allows us to reduce the problem of learning realistic locomotion into a problem of optimization through a particular objective function, for which various solutions techniques are available. We formulate a bottom-up strategy for learning multilevel muscle controllers. At the initial stage of the learning process, the animate model has no knowledge about how to contract its muscles. This is analogous to a creature which has a fully functional body, but no brain. Through repeated forward simulations with perturbed muscle activation functions, the model learns how to move most efficiently over a short time interval, by preserving those changes that improve the motion. Repeated improvements eventually produce locomotion that is very efficient.

When a sufficient level of optimality is achieved, the models abstract the low-level muscle activation functions that they have learned and associate them with the specific higher-level tasks that they are required to perform. The learning algorithm abstracts the activation functions into a highly compact representation. The representation emphasizes the natural periodicities of the derived muscle actions and makes explicit the coordination among multiple muscles that has led to effective locomotion. Finally, the synthetic creatures can put into practice the compact, efficient controllers that they have learned.

We are particularly interested in realistic motion synthesis for three dimensional creatures that are highly deformable and can locomote continuously in a 3D world. To build these models, we use mass-spring systems following Miller's approach [23]. This results in mechanical systems with a large number of degrees of freedom.

Figure 1.1 illustrates the model we used in our animation of the Sonoral coral snake (*Micruroides euryxanthus*) [11]. It serves as a good example of the issues involved in the control of deformable models. The body consists of 10 segments. Each segment

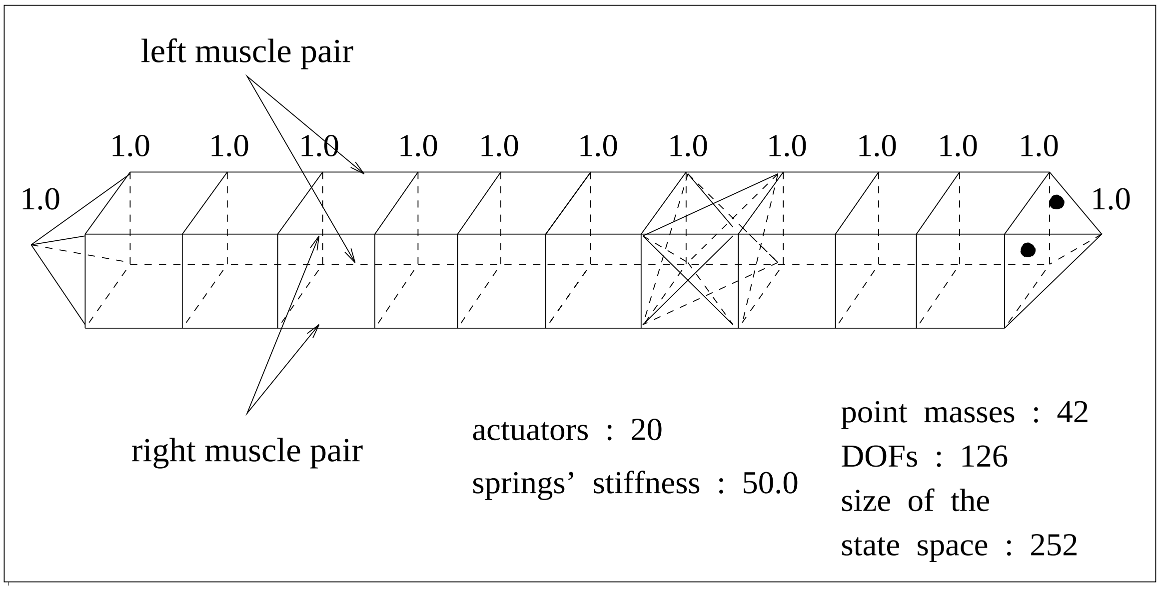

Figure 1.1: The snake model has twenty independent actuators: ten on the left side of the body and ten on the right side. Each actuator consists of a pair of muscles that share a common activation function. The numbers along the body indicate the mass of each point in the corresponding cross sectional plane. The cross-springs that maintain the structural integrity of the body are shown in one of the segments.

has a pair of muscle springs running along the sides of its body. All other springs are passive dynamic elements that maintain the structural integrity of the body. The creature can actuate the body by changing the rest lengths of the 40 springs. The muscle pairs on one side of each segment are actuated synchronously, and this yields a total of 20 actuators. Clearly, it is not particularly convinient for the animator to have direct control over each actuator. Instead the creature should learn how to control its own body. First it learns how to activate each actuator independently, then discovers the coordination needed to achieve efficient serpentine locomotion. At each learning step it performs a forward physical simulation and evaluates its progress against an objective function.

Figure 1.2 shows two frames from an animation demonstrating the low-level simulated annealing learning process (described in Section 4.2) applied to the model of the Sonoral coral snake. The snake that is farthest from the camera has learned how to locomote for the shortest period of time (only 100 annealing steps), which results in muscle control functions that are essentially random and achieve negligible locomotion. Snakes closer to the camera have learned for progressively longer periods of time, 1500 more learning steps in each case. The closest snake, which locomotes the best and wins the race, has completed 7600 learning steps.

One advantage of our approach is that it readily extends to other animate models. Another is that it allows us to equip our models with sensors that enable them to perceive their environment. Sensory perception is achieved through the objective function to be optimized. The sensory contribution to the objective function represents the degree to which a certain task has been met. When the goal changes, the model is able to sense the change. This approach allows our models to respond to dynamic events in their environment.

1.1 Thesis Organization

The thesis is organized as follows:

In Chapter 2 we review previous work in computer animation. For each approach we discuss the issue of control and reusability, as these are the key issues of the thesis.

In Chapter 3 we present an overview of our approach. We argue why it is desirable to impose a hierarchical structure on the control learning problem. We distinguish between different levels of control and discuss the issue of control abstraction.

In Chapter 4 we discuss the details of low level control learning. We define the objective function and explain the choice of optimization method.

In Chapter 5 we introduce the concept of *control window* and present its advantages in the generation of animations.

In Chapter 6 we introduce a method for abstracting the control information. We reduce the dimensionality of the representation through re-parameterization using a set of periodic basis functions. We discuss the details of the parameterization using the Fourier Transform.

In Chapter 7 we present the results obtained for several animal models. We first discuss the implementation details. Then we describe the models used to generate the results. Next we show the influence of the objective function on the control functions.

Figure 1.2: Two frames from the animation showing the learning process of Sonoral coral snake *Micruroides euryxanthus*.

Finally we give the parameterization results for the models.

In Chapter 8 we summarize and comment on the contributions of the thesis and mention possible directions for future research.

In Appendix A we discuss the details of the forward simulation using the semi-implicit Euler method.

In Appendix B we give a description of a simple spring-mass model using the script based language we designed to simplify model creation.

Chapter 2

Previous Work

The role that computers play in the production of animations has grown enormously in recent years. Traditional animation methods have been augmented with many computer based techniques that significantly improve the quality of the results. Realistic modelling and rendering using computer graphics enhances the visual impact of the animations. In the remainder of the chapter we will present the principal prior contributions to computer animation as they relate to our work.

2.1 Keyframing and Procedural Animation

Keyframing is a traditional animation technique that uses computers to calculate positions of an object in between keyframes specified by an animator. Keyframe interpolation by computer improves the efficiency, but creating animations this way can be very tedious and mechanical. Objects have no physical properties and motion realism depends absolutely on the skill of the animator. A number of methods have been introduced to increase the smoothness of generated motion. Spline interpolation of keyframes is one of the most successful techniques [17]. Despite the limitations, keyframe animation systems are still the most successful tools in the production of animations. Pixar's "Luxo, Jr." [19] is an example of what excellent results can be obtained using a computer to interpolate keyframes and render graphics models.

Procedural methods attempt to determine the kinematics of an animated object indirectly through some sort of algorithm. One of the popular methods in this class is *inverse kinematics* [3, 13]. For instance, given an articulated object and the position in space of one of its parts, this approach tries to determine the positions of the remaining parts. Systems with many of degrees of freedom are usually underconstrained and therefore difficult to handle using inverse kinematics.

2.2 Dynamic Simulation

In the dynamic simulation approach, the objects are given physical properties such as mass, inertia, energy, friction, etc. Motion is achieved by applying forces and torques to the object and integrating the equations of motion to compute velocities and positions over time [41, 26, 39, 35, 23, 4, 30, 5, 36].

The generality of this approach has allowed researchers to apply much existing work from other disciplines. Robotics and biomechanics, for example, have contributed significantly to the animation of articulated figures [38, 3, 8, 29, 2, 38, 15, 32]. The theory of elasticity and the finite element method have been used in animating flexible objects in simulated physical environments [22, 34, 35, 30].

Dynamic simulation may involve complex sets of differential equations that may be computationally intensive. Relative lack of control can be a severe impediment to the application of dynamic simulation to animation. *Inverse Dynamics* has been introduced to control dynamic graphics models [14, 15, 5]. Given a set of kinematic constraints this approach tries to determine the forces which would result in the desired, constrained behavior.

2.3 Optimal Control

Optimal control methods attempt to synthesize motion based on certain optimization criteria. Brotman and Netravali [7], for example, assume that the most natural looking motion results from minimal and continuous driving forces. The problem is usually formulated through time t in terms of an objective functional

$$E = \int_{t_0}^{t_1} F(t)dt$$

which must be minimized subject to equations of motion expressed as constraints [40]. $F(t)$ is an energy function that depends on the parameters we wish to control. Changing the objective function allows us to generate different motions. Witkin and Kass [40] produce interesting effects by changing the goals of the motion and the physical properties of the model. The difficulty in formulating suitable objective functions can be a severe limitation of their spacetime constraints as is the fact that it treats physics as a penalty term (that can be stretched like a spring). The optimization process is performed on the entire state and actuation spaces of the models. This limits the complexity of the models that can be handled and the length of the animations that can be produced. The latter problem has been addressed by Cohen in [10]. His *Spacetime Windows* system breaks down the long animation into a sequence of smaller pieces, as we do in our work.

2.4 Self-Controlling Systems

A different approach toward the synthesis of computer animation, and the one that we pursue, has been presented in [27, 25, 24]. In this work, articulated figures are given the capability of finding their preferred modes of locomotion. The use of a generate-and-test strategy through repeated forward simulation eliminates the need to incorporate the equations of motion directly into the objective function as in the previous methods. Park *et al.* [27] propose an approach to control that is related to our work. They search in the actuator space for good solutions, but they do not perform global optimization to find the best solution. Such an approach is likely to fail on any model with a large number of actuators because it will become trapped in local suboptima. Van de Panne and Fiume's [25] models are equipped with simple sensors that probe the environment and use the incoming information to influence the control decisions. A similarity between their work and ours is the use of simulated annealing as a global optimization procedure to avoid local suboptima. Ngo and Marks' [24] stimulus-response control algorithm presents a very similar approach. The genetic algorithm is used to find control solutions.

The issue of realism has not been addressed in the two latter works. Furthermore, no prior graphics work addresses the difficulties of unconstrained three-dimensional motion of animals. Three-dimensional animations are harder to create because the complexity of the problem grows. The state space increases on average threefold, the structure of the models quite often needs to be augmented so that the three dimensional motion can be accomplished, there tend to be more control decisions that must be taken into account, etc. The models in [24], for instance, are mostly planar configurations of rigid links (although the authors presented a creature in their talk that twisted its torso during the animation), which make no serious attempt to model any real animals.

2.5 Locomotion Coordination

The coordination problem involves the organization of movements of a complicated figure with many degrees of freedom into a coherent motion. The problem can be reduced to discovering those degrees of freedom which may perform a given motion and to finding the temporal interdependencies between them. McKenna and Zeltzer [20] propose the use of biologically based mechanisms to coordinate the complex patterns of movement of a six-legged insect. They create a hierarchical control structure whose top level, called *a gait controller*, is responsible for coordinating the separate limbs. The middle level consists of dynamic motor programs responsible for delivering forces to create the movements. The bottom level is a system for simulating the forward dynamics of articulated figures (Figure 2.1). Our approach differs from theirs since we are trying to abstract the higher level coordination mechanisms from low level control patterns.

Raibert and Hodgins [32] use a control system handcrafted to perform certain simple behaviours to animate dynamic legged locomotion. Control algorithms regulate the running speed, maintain balance, preserve posture, etc. Finite state machines are implemented to track the ongoing behaviour of the model and to synchronize the basic control actions to produce a set of gaits: biped running and galloping; quadruped trotting, bounding, and galloping; and kangaroo hopping.

Brooks [6] demonstrates distributed control systems for mobile robots. Such systems are built incrementally with different control mechanisms at each level of the construction. He shows how basic behaviors, such as walking, can emerge from a network of simple reflexes with little centralized control.

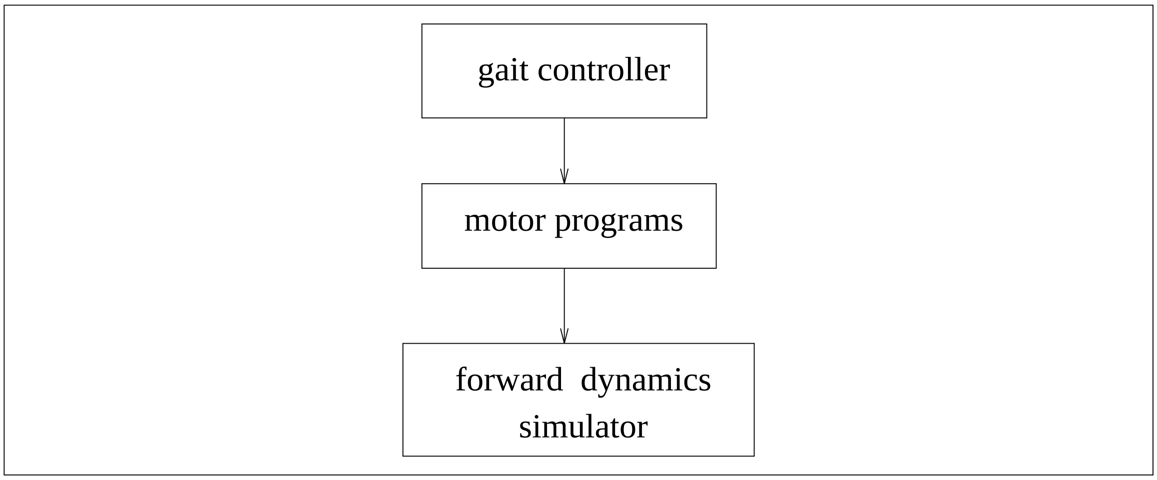

Figure 2.1: Hierarchical System for Locomotion Coordination.

No prior work addresses seriously the problem of motion synthesis and control for deformable animal models. In the case of the snakes and worms of Miller [23] or the fish of Tu *et al.* [37], the controllers responsible for the locomotion have been carefully hand crafted using knowledge gleaned from the biomechanics literature [21] and long hours of experimentation. This thesis develops algorithms that can automatically learn such controllers from first principles.

Chapter 3

Overview of Our Approach

3.1 Control

A limitation of most of the research in the area of motion synthesis is that it concentrates mainly on the aspect of low level control. Choosing to search over the actuator space as opposed to the state space does not always make the control problem sufficiently simple. Some animate models may have many actuators, but the control structure producing motion might be quite simple. Consider for example trying to make a caterpillar walk. Treating each leg independently complicates matters. We believe that an important component of motion control synthesis is to reduce the high-dimensional actuator spaces of the animate model into task-specific state spaces of very low dimension. This is a problem of control abstraction; that is, of learning a highly compact representation that exploits the natural periodicities of the derived muscle actions and makes explicit the coordination among multiple muscles that leads to effective locomotion.

We will distinguish between two different levels of interest. The first level will be concerned with issues of time-localized control. Over a particular interval of time, we will seek a set of muscle controls that produces the desired configuration of an animate model at the end of the interval. In the remainder of the thesis we will refer to this as *the low level control problem.* The second level deals with the control issue on a more global scale. The time of interest spans the whole or a significant part of the animation. The control problems may be defined in terms of complex goals that should be met. We will refer to this as *the high level control problem.*

3.2 Reusability

A hierarchical approach to control leads naturally to reusable solutions. Finding a solution to the problem of low level control is a relatively time consuming process. It is therefore essential that we be able to reuse earlier results in the creation of new animations. For this reason we search for good low level control solutions for a set of simple tasks, which tend to be costly, and use them as building blocks to achieve higher level goals. Once a solution to a low level control problem is found, it is abstracted in order to reduce the dimensionality of the controller. The advantage of this is twofold. First, abstraction results in a compact representation that admits a direct interpretation. Second, the reduced dimensionality of the control problem makes it much easier to generalize the results across different tasks.

3.3 Operating Modes

The layered structure of the control problem leads to two different operating modes for our system. One is what we call *a learning mode.* In this mode the system is configured to learn a solution to a simple task by perturbing the control functions until a good set of controls is found (Figure 3.1A). The other mode, referred to as *the execution mode*, uses the controlers abstracted for some simple tasks during the learning process as building blocks to create animation. At this stage the program is trying to find a sequence of abstractions that would best approximate the goals specified by the animation (Figure 3.1B).

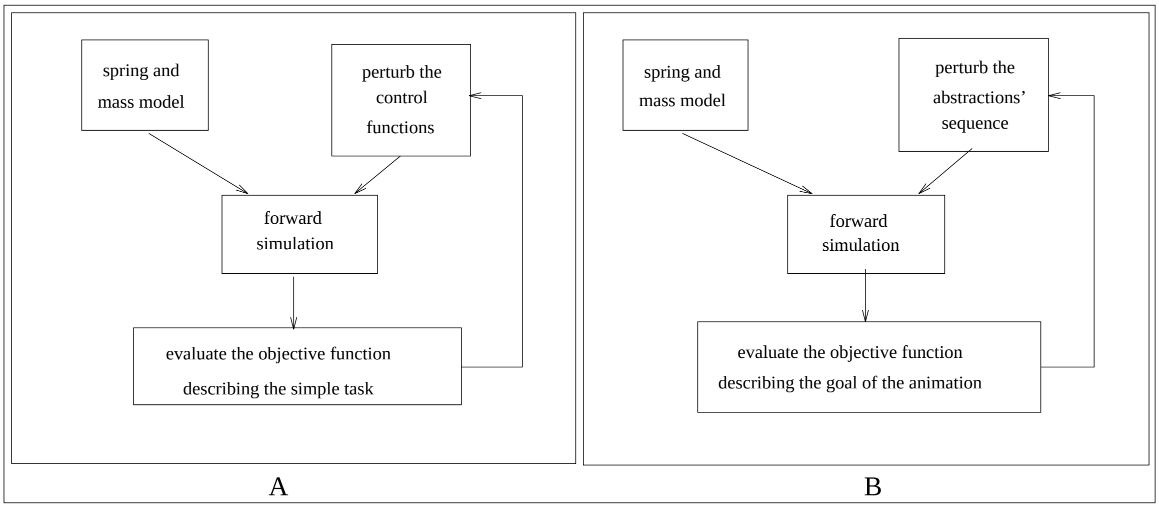

Figure 3.1: In the learning mode (A), the algorithm tries to find a set of control functions that perform a simple task. In the execution mode (B), it tries to find a sequence of abstractions that best approximate the goals specified in the animation.

Chapter 4

Learning Low Level Control

Our low-level learning technique generates a potentially large number of actuator control functions. Each of these functions is then used to drive a short-time forward simulation of the physical model (see Appendix A). The result of each simulation is a motion that is evaluated against an objective function that serves as a quantitative measure of optimality. To obtain control functions that produce the desired motions, we must sample a large number of candidate control functions and perform many forward simulations to evaluate the optimality of these control functions; hence, low-level learning is slow. The advantage is that it works well.

4.1 Objective Function

To recognize control functions that produce good results, we need to introduce an evaluation metric. The distinction between good and bad motions depends largely on the goals that the creature must accomplish. Since we want the creatures to control their own low-level motions, the low-level control is intended to be autonomous. That is, it should be based on first principles and not subject to detailed specification by the animator. Regardless of the complexity of the tasks our model is able to perform and how independent it is in its ability to simulate natural behavior, we would like to exercise some authority over certain qualities of the control that might influence, say, the smoothness of the generated motion. For that reason we include evaluation criteria on the control function itself. Our objective function is the weighted sum of a term that evaluates the control function and a term that evaluates the motion synthesized by the control function. Figure 4.1 illustrates this schematically.

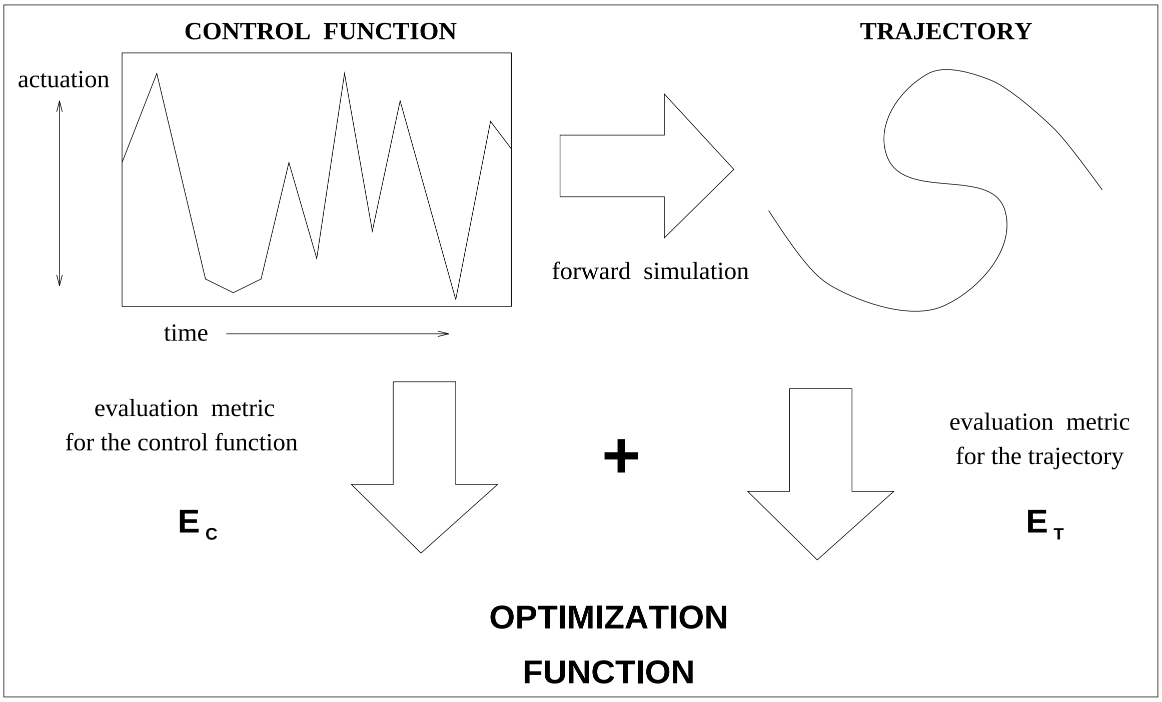

Figure 4.1: The objective function guiding the optimization is a weighted sum of evaluation metrics for the trajectory and for the control function.

In our formulation, the objective is to maximize the functional [1]

$$I(\mathbf{u}(t)) = \int_{t_0}^{t_1} [\mu_1 E_C(\mathbf{u}(t)) + \mu_2 E_T(\mathbf{v}(t, \mathbf{u}(t)))] \, dt \tag{4.1}$$

where

[1] We could easily perform a minimization instead of maximization. To do that we would need to change the sign of $I(\mathbf{u}(t))$.

$\mathbf{u}(t)$	–	vector of control functions
$\mathbf{v}(t, \mathbf{u}(t))$	–	motion trajectory obtained through forward simulation using $\mathbf{u}(t)$
E_T	–	term associated with the trajectory
E_C	–	term associated with the control function
μ_1, μ_2	–	weighting variables.

In our learning experiments, the most frequently used criterion for a good trajectory is the distance to the goal. Other possible criteria are: How well does the creature match a certain speed? How far does it move away from the goal? How closely does it match its distance from a certain point? etc.

We usually wish to influence the smoothness of the control functions. The rationale behind this is that energy-efficient actuations are usually smooth actuations. We use two functions to encourage smoothness:

$$E_{C_1}(\mathbf{u}(t)) = -\frac{1}{2}{\mathbf{u}_t}^2(t) \tag{4.2}$$

and

$$E_{C_2}(\mathbf{u}(t)) = -\frac{1}{2}{\mathbf{u}_{tt}}^2(t), \tag{4.3}$$

where $\mathbf{u}_t$ is a vector of first order time derivatives of control functions, and $\mathbf{u}_{tt}$ is a vector of second order time derivatives. These terms are potential energy densities of linear and cubic splines in time respectively [1].

Smoothness constraints have been incorporated into the objective function in the past in order to make the models conform to certain quality of motion. Witkin and Kass [40] use a range of constraints depending on the effect they want to achieve, without giving very many specific examples. Brotman and Netravali [7] claim that visually the best motion is achieved when the amount of control applied is minimal and the motion is 'not rough'. By roughness they mean the changes in the velocity of the animated objects. They minimize the functional

$$\mathbf{J} = \int_{t_0}^{t_1} [\alpha\|\mathbf{v}_t(t)\|^2 + \beta\|\mathbf{\Omega}_t(t)\|^2 + \gamma\|\mathbf{u}(t)\|^2]dt \tag{4.4}$$

where

$\|\mathbf{v}(t)\|, \|\mathbf{\Omega}(t)\|$	–	measure of roughness of linear and angular motion.
$\|\mathbf{u}(t)\|$	–	measure of control.

Girard [12] minimizes jerk

$$J(\mathbf{v}(t)) = -\frac{1}{2}\int_{t_0}^{t_1} \mathbf{v}_{ttt}{}^2(t)dt \tag{4.5}$$

for his animations of hand motion. This approach concentrates specifically on the trajectory of motion and neglects the impact that the control has on the resulting motion.

4.2 Optimization

We have chosen simulated annealing [31] for the optimization of our objective function. This method has a number of features that make it particularly suitable to our application. It is applicable to problems with a large number of variables—search spaces large enough to make exhaustive search prohibitively expensive. Simulated annealing does not require gradient information about the objective function I. Analytic gradients are not available in our case, since it does not seem feasible to calculate the gradient analytically given the complexity of our models. We have not explored the possibilities of computing the gradient numerically, but this information can be useful only if we can show that ascent methods are applicable to our problem. Ascent algorithms terminate in a local maximum and the obtained local maximum depends on the initial configuration. Van Laarhoven and Aarts [18] give a number of heuristics that can improve the main drawbacks of what they call *iterative improvement* algorithms. One of them is the execution of the algorithm for a large number N of initial conditions. As N approaches infinity such an algorithm finds a global maximum with probability one. Solutions obtained by simulated annealing do not depend on the initial configuration and give a solution that is usually close to the global optimum [18]. Robustness against local suboptima is important in obtaining control functions that produce realistic motion. Section 7.3 presents a quantitative evaluation of different optimization techniques.

Simulated annealing may be applied after discretizing the actuator control functions $\mathbf{u}(t)$ in time to obtain the set of discrete *control points* $\mathbf{u}_i = \mathbf{u}(t_i)$ for $0 \leq i \leq N-1$. The algorithm randomly perturbs the current values of the $\mathbf{u}_i$ to obtain new values. It avoids getting locked into local suboptima by accepting the new $\mathbf{u}_i$ so long as it increases the objective I, but occasionally selecting $\mathbf{u}_i$ that temporarily decrease I. More formally, if the change ΔI in the objective function I is positive, then the new $\mathbf{u}_i$ are accepted with probability $P = 1$, but if $\Delta I < 0$, then the new $\mathbf{u}_i$ are accepted with probability $P = \exp(\Delta I / kT)$, where k is Boltzmann's constant and T is the temperature. The temperature is decreased gradually according to an annealing schedule. When the value of T is large, backwards steps are taken quite freely. For small values of T mainly perturbations causing small positive changes in the value of the objective function are accepted. This is reasonable, since at this point we are sufficiently close to the optimum and we want to refine the final solution.

Chapter 5

Short-Time Control Windows

Learning low level control involves using the simulated annealing algorithm to iteratively update a set of control points u_i in order to produce increasingly better locomotion. To reconstruct the continuous control functions $\mathbf{u}(t)$, we interpolate the set of evenly spaced control points. Linear interpolation yields satisfactory results. In our application, the functions $\mathbf{u}(t)$ represent actuator activation functions which control individual muscles in the creature. [1] The forces generated using piecewise linear control functions are still C^2 continuous. Using higher order methods for function modeling leads to increased smoothness of the generated control functions. We have noticed however that this does not have a significant impact on the smoothness of the generated motion. To reduce the computational complexity of the learning process while permitting the use of perceptual input to guide the motion, we divide the total animation time into a sequence of shorter intervals that we call *control windows*. We find control solutions within each window by performing sequential global optimizations.

Figure 5.1 shows a control function and a set of control windows used to describe it. Consecutive windows overlap to ensure the global continuity of the control function over all the windows. Since we use linear interpolation between control points to obtain control functions, only a one point overlap is required. This means that the

[1] It is straightforward to use different control parameters for different models, such as an angle between two joints, a force acting on an object, etc.

first control point of the $i+1$'st control window is constrained to the same value as the last control point of the i'th window. The bottom of the figure shows the control windows separated, each one covering its own interval of the control function. This stresses the fact that finding the solution for each control window is a separate optimization problem.

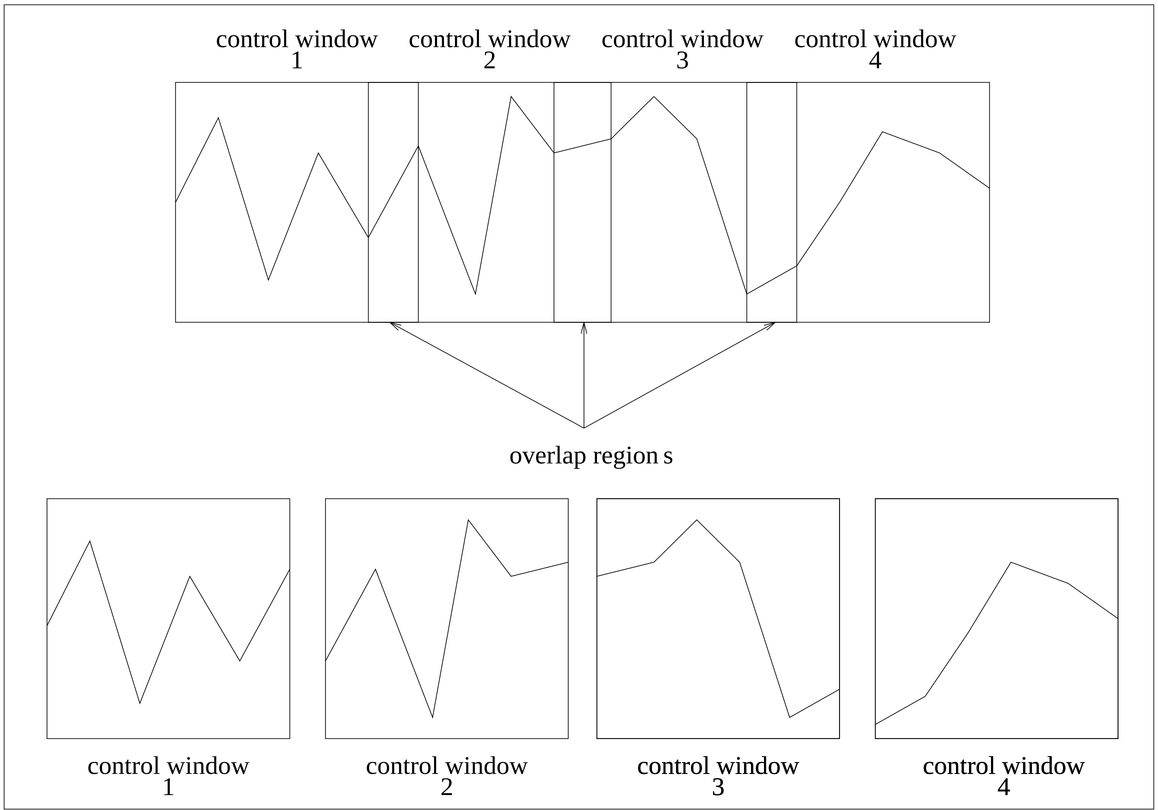

Figure 5.1: Consecutive control windows overlap to ensure the continuity of the control function. The control function is a piecewise linear interpolation of control points. The bottom of the figure shows the control windows separated, each one covering its own interval of the control function.

The control window approach to generating animation has one more interesting advantage. It focuses the low-level learning process to the immediate goal to be satisfied. Long term strategy is not immediately relevant to the task of learning effective general-purpose locomotion patterns, such as moving straight and turning. It can, in fact, be a distraction that may retard the learning rate. The resulting control sequence will yield only globally suboptimal solutions (and potentially very suboptimal ones) but this is in general desired, since animals rarely try to optimize their behavior on a global scale.

Short term optimization is useful in situations where the goal to be met by a creature changes over time. Cohen [10] gives a good example of this in an animation of one creature chasing another. Limiting the time over which the optimization is performed allows the creatures to modify the chase-escape strategy based on the updated information coming from the solution produced by the earlier parts of the animation. Figure 5.2 shows a likely outcome if the optimization were to be performed on the full animation sequence (A), and the results that could be accomplished using the control windows to produce the animation (B). The second choice gives a much more realistic solution. The markers on the curve show the points where the control had been switched to a new control window and updated sensory information about the location of the creature has been assimilated.

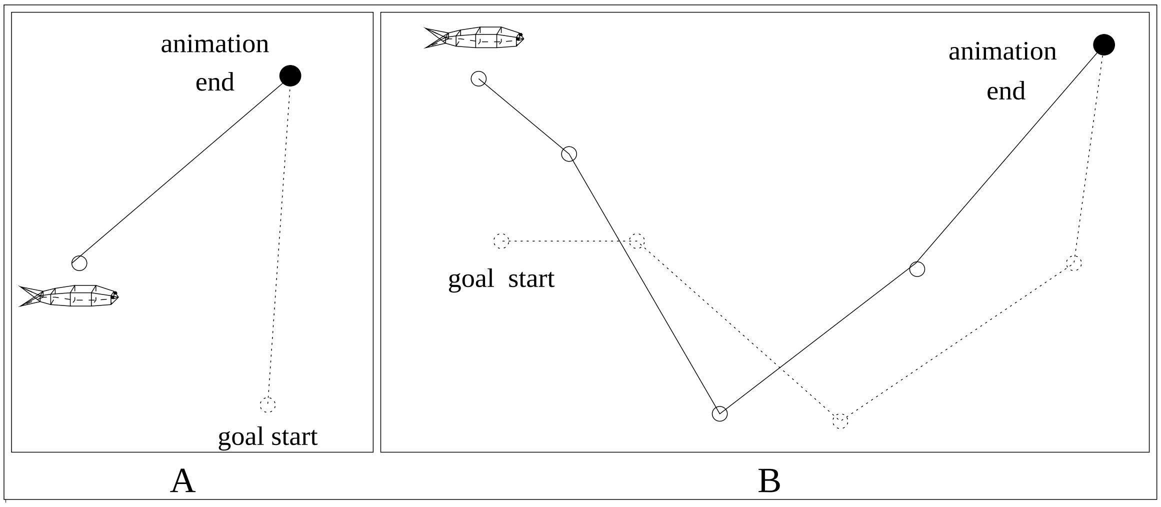

Figure 5.2: Using control windows to compute trajectories results in much more realistic scenarios for a chase animation (B). The dotted line shows the trajectory of the animal being chased. The solid line indicates the trajectory of the predator. The markers on the curve show the points where the control had been switched to a new control window and updated sensory information about the location of the creature has been assimilated.

As a final point regarding low level control learning, we would like to stress that there is no evidence that any natural motion is optimal in the sense of our objective function. Statements that any natural locomotion must be absolutely optimal in some specific sense are probably untenable. The empirical results from our experimants produce suboptimal, though natural looking locomotion.

Chapter 6

Abstraction to Higher-Level Controllers

Learning a good solution over a control window is a time consuming process, because of the high dimensionality of the problem due to numerous control variables, the difficulty in obtaining analytic gradient information to accelerate the optimization of the objective function, and the presence of suboptimal traps that must be avoided. Consequently, it does not seem feasible to produce animation by perpetually recomputing new control windows. The learning procedure must be able to abstract higher level controllers from the low level controllers that it has generated initially, and apply these controllers in future situations.

The process of abstraction takes the form of a dimensionality reducing change of representation. More specifically, it seeks to compress the information content of the many control points to a compact form in terms of a handful of simple basis functions. In mathematical terms, it tries to represent the control functions as accurately as possible as a linear combination of a small set of temporal basis functions $\mathbf{B}_i$; i.e.,

$$\mathbf{u}(t) = \sum_{i=1}^{M} \alpha_i \mathbf{B}_i \tag{6.1}$$

where the α_i are scalar quantities and M is a small number.

Natural locomotion patterns are often periodic [28], and this constrains the choice of a suitable basis set. A natural choice is to use globally periodic basis functions; that is to say, the trigonometric Fourier basis. We employ Fourier analysis to perform the change in basis. Theoretically the change in basis does not modify the dimensionality of the representation. However, if the Fourier space is a suitable representation that captures the temporal structure of the control functions, the dimensionality reduction can be achieved trivially by eliminating all basis functions whose coefficients α_i in equation 6.1 are negligible. This will result in a small set of M significant basis functions with associated coefficients that constitute the abstracted controller.

6.1 Fourier Representation

If N is the number of the points generating the control function then we approximate the control function by a sum of cosines:

$$\mathbf{u}(t) = \mathbf{U}_0 + 2\sum_{k=1}^{N/2} \|\mathbf{U}_k\| \cos(2\pi f_k t + \varphi_k). \tag{6.2}$$

We use the Fast Fourier Transform (FFT) [31] to generate the basis representation.

If the modeled function is fairly smooth and does not have discontinuities, most of the terms in 6.2 will be negligible and we can get a compact representation by using only the FFT coefficients with the largest amplitudes. Figure 6.1 illustrates this process. We keep only those frequency modes that exceed a certain threshold and annihilate the rest.

The model learns through simple task abstraction how it should vary the reduced parameter set of the abstracted controller to produce desired actions, such as executing turns. It can subsequently put these learned abstractions into practice to accomplish higher level tasks, such as target tracking.

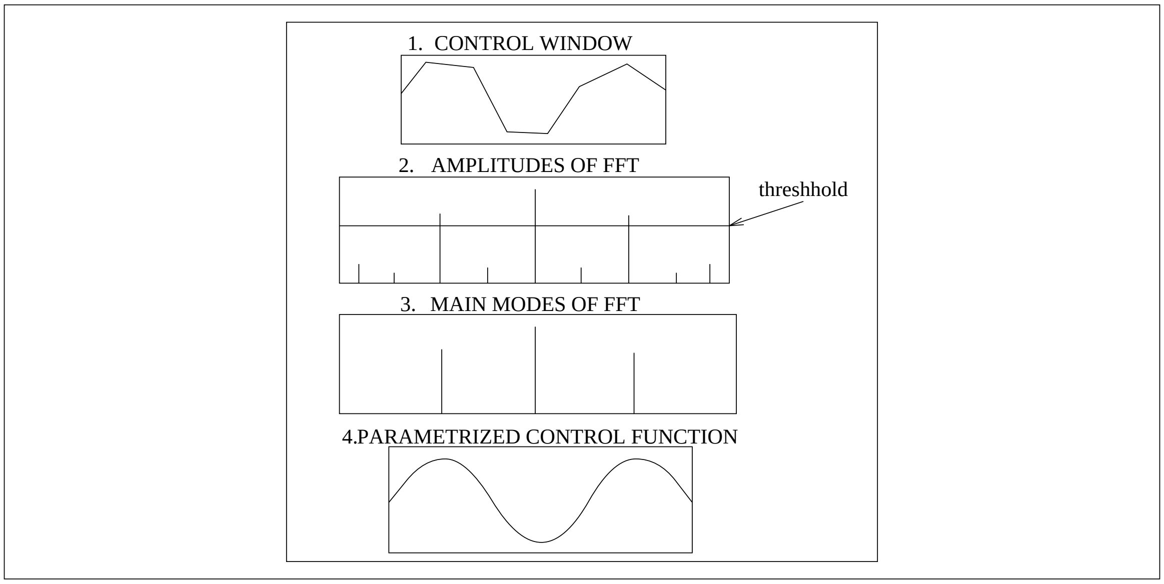

Figure 6.1: Only the FFT modes above the threshold line are used in the final parameterization of the control function.

6.2 Fast Fourier Transform

Given a certain function of time $h(t)$, we can represent it as a function of frequencies $H(f)$. The relationship between the two representations can be described by means of the Fourier Transform

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{2\pi i f t} dt. \tag{6.3}$$

In a discrete case, where the function in time domain is described by a set of N evenly sampled points with a sampling interval Δ

$$h_k = h(k\Delta) \qquad k = 0, 1, \ldots, N-1,$$

we estimate the FFT for discrete frequency values

$$f_n = \frac{n}{N\Delta} \qquad n = -\frac{N}{2}, \ldots, \frac{N}{2}.$$

In such a case, the integral (6.3) can be represented as a discrete sum

$$H(f_n) = \Delta \sum_{k=0}^{N-1} h_k e^{\frac{2\pi i k n}{N}}$$

If we use the notation

$$H_n = \sum_{k=0}^{N-1} H_k e^{\frac{2\pi i k n}{N}}$$

then the discrete Fourier transform maps N complex numbers (the h_k's) into N complex numbers (the H_n's). Since in our application h_k's are real, then $H_{-n} = H_n^*$ and

$$\begin{aligned} H_{-n} + H_n &= \|H_n\|(\cos\varphi_n + i\sin\varphi_n) \\ &+ \|H_n\|(\cos\varphi_n - i\sin\varphi_n) \\ &= 2\|H_n\|\cos\varphi_n \end{aligned}$$

We can use this fact to represent our control functions in terms of a sum of cosines:

$$f(t) = H_0 + 2\sum_{k=0}^{N/2} \|h_k\| \cos(2\pi f_k t + \varphi_k)$$

Chapter 7

Results

In this chapter, we present results obtained for several animal models. First, we describe the models used to generate the results. Second, we give the implementation details. Third, we compare the optimization results obtained using different teqniques. Fourth, we show how changes in the objective function influence the learned control functions. We conclude by showing examples of abstract controls for three different models: a fish, a snake and a stingray.

7.1 Description of Animate Models

We use spring-mass animal models in our work. Appendix A reviews the mechanics of spring-mass systems. Creatures such as snakes, worms, fishes and many other sea animals, whose entire bodies deform under the action of internal muscles, and for which the external structure is relatively simple, are well suited to spring-mass modeling. Appendix B gives a description of a simple spring-mass model using a script based language that we designed to simplify the process of model creation.

7.1.1 Snake

Figure 7.1 shows the model of the snake. All the muscles were allowed to change their default length by thirty per cent. All the springs have the same stiffness. Mass is distributed evenly between all points. In modeling snake locomotion, we use directional friction in the form introduced by Miller [23] to compute the ground forces for the snake. In this formulation, friction changes depending on the direction in which a mass particle is moving. The local forward spine vector $\mathbf{s}_i$ defines an orientation of the snake's i'th segment:

$$\mathbf{s}_i = \frac{\mathbf{x}_{i+1} - \mathbf{x}_i}{\mid \mathbf{x}_{i+1} - \mathbf{x}_i \mid}$$

where $\mathbf{x}_i$ is the center of mass of the i'th segment. We damp the tangential component of velocity with respect to the local forward unit vector by applying force in the direction $\mathbf{s}(\mathbf{s} \cdot \mathbf{v}) - \mathbf{v}$. To prevent the particle from sliding backward we apply a force proportional to $-\mathbf{s}(\mathbf{s} \cdot \mathbf{v})$ (Figure 7.2B).

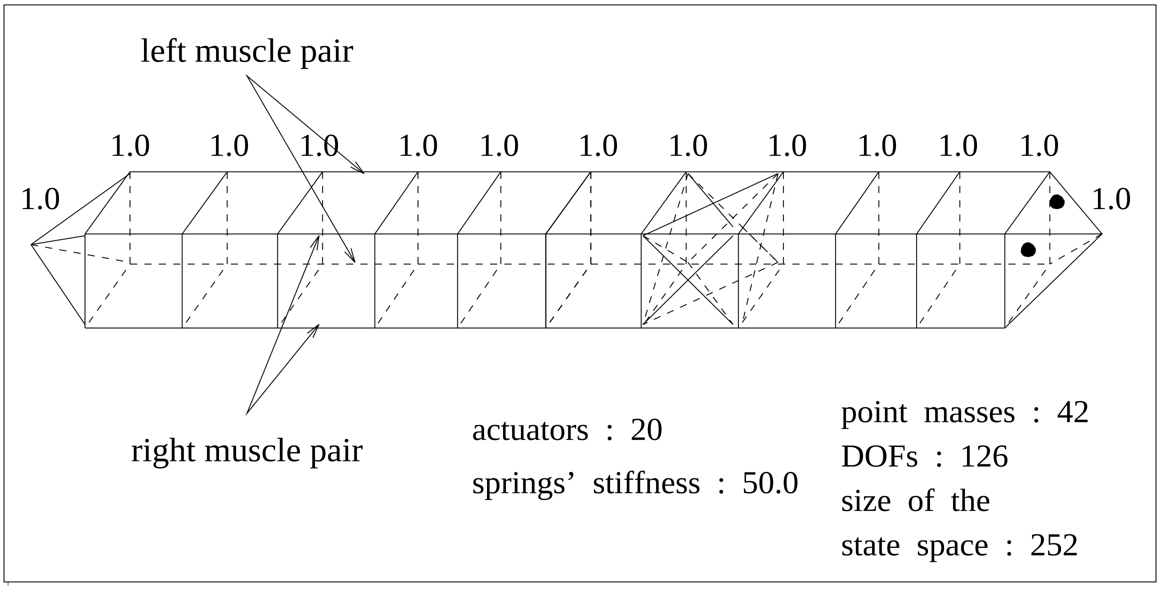

Figure 7.1: The snake model has twenty independent actuators: ten on the left side of the body and ten on the right side. Each actuator consists of a pair of muscles that share a common activation function. The numbers along the body indicate the mass of each point in the corresponding cross sectional plane. The cross-springs that maintain the structural integrity of the body are shown in one of the segments.

7.1.2 Aquatic Animals

For aquatic animals we compute the water forces acting on each of the model's faces. If the dot product of the vector normal to the face and the velocity of the face is greater then zero (Figure 7.3A), then water resistance is calculated according to the

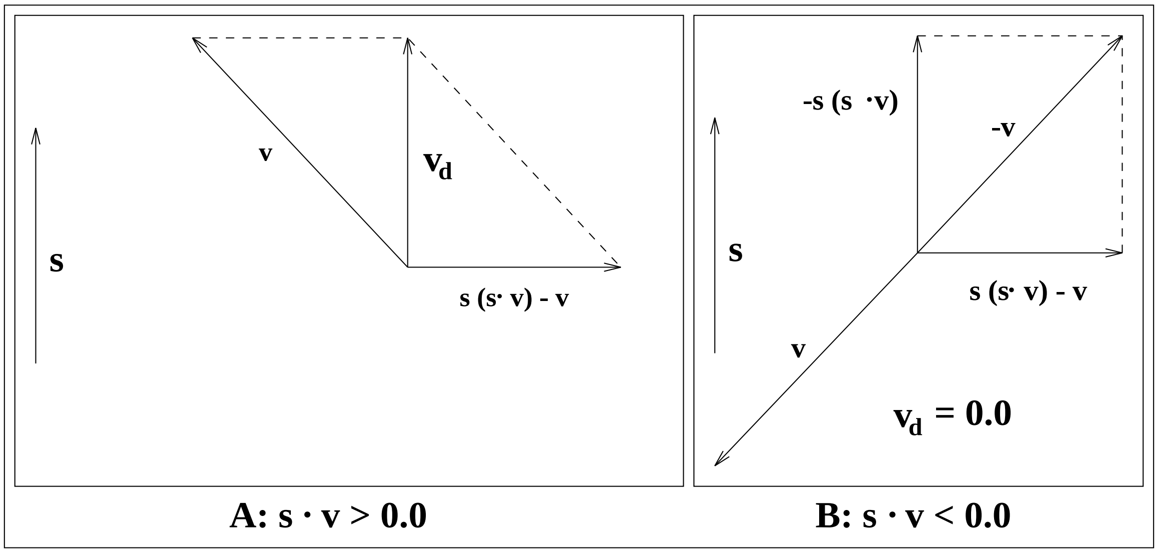

Figure 7.2: Two possible cases for the computation of directional friction force. $\mathbf{v}_d$ indicates the velocity of the point mass after applying the directional friction forces.

formula

$$\mathbf{F}_w = c_w f_a \mathbf{n}(\mathbf{n} \cdot \mathbf{v}),$$

where c_w is the water resistance constant and f_a is the area of the face over which we compute the force. This corresponds to the situation when the face is pushing against the water. In all other cases we assume that $\mathbf{F}_w = 0$ (Figure 7.3B).

Figure 7.4 shows the model of the Leopard shark [9]. The back muscles can change their natural length by ten per cent, all the other muscles by twenty per cent. The points in the center of the body have five times as much mass as the other points. The points on the tail of the fish have five times lighter than other points. Figure 7.5 shows the model the Kuhl's stingray [33]. All the muscles can change their default length by twenty per cent. Mass is distributed evenly between all points. All the springs have the same stiffness for both models.

7.2 Control Window and Annealing

Each control window contains 15 N-dimensional control points, where N is the number of actuators in the model. For each vector-valued control function the control

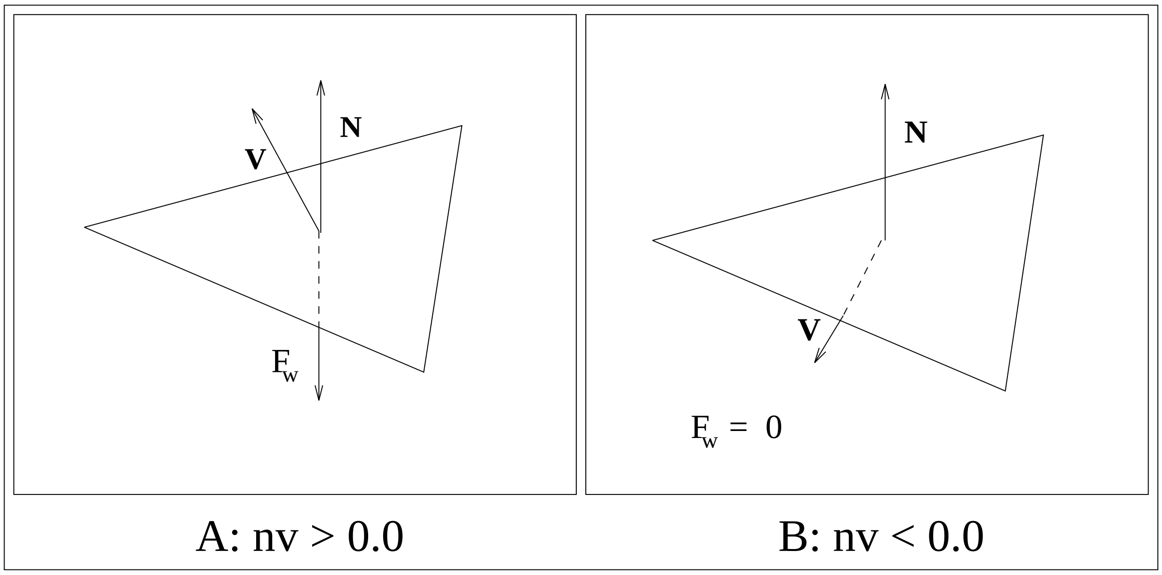

Figure 7.3: Two possible cases for the computation of water force.

points are spaced every $50\Delta t$ sec, where $\Delta t = 0.055sec$. Each window spans, therefore, exactly $15 \times 50 \times \Delta t = 4.13sec.$ of the animation.

We start the simulated annealing procedure at the temperature $T_0 = 0.5$. Each new temperature $T_{i+1} = 0.9T_i$. The maximum number of steps before the temperature drop is $10NM$ and the maximum number of accepted perturbations before the temperature drop is NM, where M is the number of functions. For $N = 15$ it takes about one hour on an SGI Indigo workstation to get a solution for each control function. At each step, values of all control points are perturbed randomly. The maximum acceptable change of each perturbation is equal to ten per cent of the range of admissible values. So, for example, if a muscle is allowed to change its rest length by no more then twenty per cent, at each reconfiguration the change in the muscle length will be at most two per cent. The bound on the change of control point value remains fixed for the whole annealing schedule. This results in much faster convergence rate compared with the situation where we decrease the bound together with the temperature.

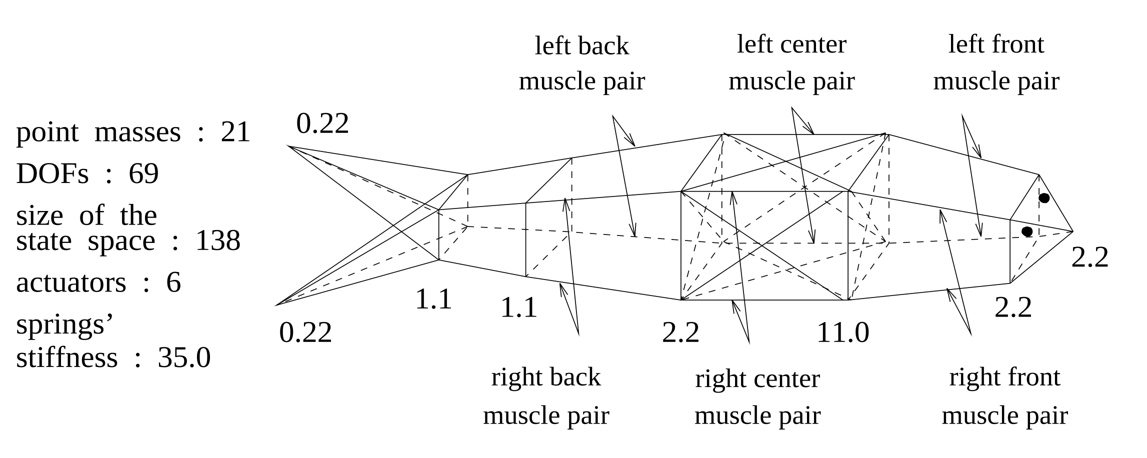

Figure 7.4: The fish model. Each of the six actuators indicated in the figure consists of a pair of muscles that share the same activation function. The numbers along the body indicate the mass of each point in the corresponding cross sectional plane. The cross-springs that maintain the structural integrity of the body are indicated in one of the segments only.

7.3 Different Optimization Strategies

Figure 7.6 shows the muscle activation functions obtained for the straight swim of the fish model using different optimization strategies. Ascent strategies were simulated using the annealing algorithm with the initial temperature $T_0 = 0.0$ and different values for the maximum permissible perturbation at each step of the method. This ensures that our algorithm always chooses only the configurations that move us closer to the maximum. This method can be thought of as only an approximation to an ascent algorithm since we will still be able to avoid some local maxima. Since we perturb our control functions randomly at every step however, there is no concept of a direction of ascent in this solution. The algorithm will be able to cross local maxima of size less then the maximum size of the perturbation. If we choose the maximum permissible perturbation sufficiently small, such crosses should occur infrequently. Figure 7.6A shows the results obtained using the ascent algorithm for a single initial configuration obtained using annealing with $T_0 = 0.0$ and with $P = 0.02$, where P is the maximum allowed perturbation. The control functions obtained are very close to the control functions obtained using simulated annealing (Figure 7.6B). This may imply that our objective function does not have many local maxima and that it may

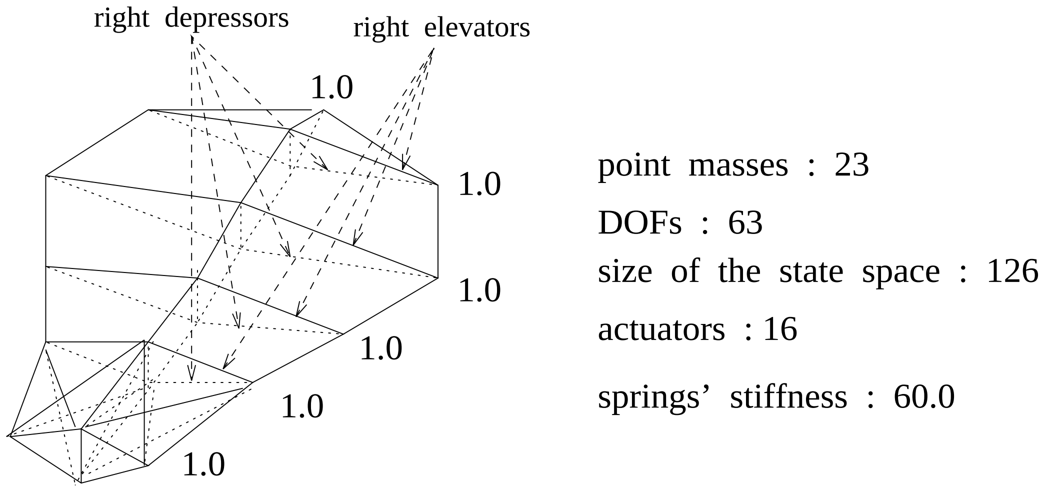

Figure 7.5: The ray model has four sets of actuators: left and right depressors and left and right elevators. The numbers along the body indicate the mass of each point in the corresponding cross sectional plane. The cross-springs that maintain the structural integrity of the body are indicated in one of the segments only.

in fact be monotonic. Figure 7.6C shows the results obtained using the random search obtained using the annealing code with $T_0 = 0.0$ and with $P = 0.5$. Large value of P ensures the randomness of each reconfiguration. The resulting control functions produce motion that is jerky and crude. Clearly the method failed to converge to the optimum of the objective function. The reason for this may be that the search space of our problems is very large and therefore random reconfigurations make higher energy states harder to find with every step.

To learn more about the topography of the objective function we conducted an experiment in which we compute the value of the objective function along hyperlines in randomly chosen directions radiating from the maximum obtained using the simulated annealing method. Figure 7.8 shows the behaviour of the objective function for the fish model. Figure 7.9 shows the behaviour of the same objective function for the snake model. In both cases the function is surprisingly well behaved. Moving from the maximum in any direction decreases the value of the function, but there are minor local maxima along the way.

For Figure 7.7 we move along a diagonal line in control point space starting at the global maximum to the point that is symmetric with respect to the origin in the control point space. By the symmetry of our muscle models this is also a global maximum, as is clearly visible from the figure. We obtain a symmetric function with a minimum centered at the origin and two global maxima at the boundaries of the line.

The above experiments indicate that our objective functions are smooth and are monotonically increasing along lines leaving the origin. This conclusion is true if we assume that control windows are of length not bigger then the one used for the experiment or that the model is not able to move away from the goal. If we make the size of the control window arbitrary large, the functions will remain smooth and monotonic, but the global minimum will move away from the origin.

7.4 Influence of Objective Function on Control Functions

In Section 4.1 we discussed how including into the objective function a term that evaluates the control functions allows us to retain some influence on the resulting motion. The extra term allows us to control the smoothness of the simulated motion, velocity, or energy spent in moving. We can produce a wide range of simple but important effects that enhance the realism of our simulations.

For all the results in this section the objective was to get as close to the goal as possible, but in each case we add a different term to evaluate the control function. The second term of the objective functional (4.1) which is associated with the learned motion is defined as

$$E_T = -||\mathbf{d}_g||^2,$$

where $\mathbf{d}_g$ is the distance from the goal. The first term, E_c is associated with the control function. Figure 7.10A shows the results obtained for the discrete smoothness term (equation 4.2) given by

$$E_{C_1} = \frac{1}{2h^2}(\mathbf{u}_{i+1} - \mathbf{u}_i)^2$$

where h is the timestep between the consecutive control points. The objective function was evaluated for three different values of the weighting factor μ: 0.0, 0.1, 0.2 (top to bottom). As the value of μ increases, both the amplitude and the frequency of the learned control functions drop.

Figure 7.10B shows the results obtained with the discrete version of the energy term (equation 4.3):

$$E_{C_2} = \frac{1}{2h^4}(\mathbf{u}_{i+1} - 2\mathbf{u}_i - \mathbf{u}_{i-1})^2$$

The objective function was evaluated for three different values of the weighting factor μ: 0.0, 0.002, 0.006 (top to bottom). As the value of μ increases, the amplitude of the learned control functions remains constant and only the frequency drops.

7.5 Learning Low Level Controls

Figure 7.11 illustrates the control functions learned for the motion straight task for three different animals: snake(A), ray(B), fish(C). For the snake model and the fish model the dotted lines indicate the muscle actuation functions for the left side of the body. The solid lines indicate the muscle actuation functions for the right side of the body. For the ray model the dotted lines indicate the actuation functions for the elevator muscles. The solid lines indicate the actuation functions for the depressor muscles.

Figure 7.13 shows two frames from an animation which demonstrates the low-level simulated annealing learning process described in Section 4.2. The process is applied to the model of a leopard shark (*Triakis semifasciata*). A race is staged between six shark models. The shark that is farthest from the camera has learned how to locomote for the shortest period of time (only 90 annealing steps), which results in muscle control functions that are essentially random and achieve negligible locomotion. Sharks closer to the camera have learned for progressively longer periods of time, 1350 more annealing steps in each case. The closest shark, which locomotes the best and wins the race, has completed 6840 steps.

Figure 1.2 shows two frames from an animation of a scenario similar to the previous one, with similar results. The learning process is applied to the model of a coral snake (*Micruroides euryxanthus).* The model closest to the camera has completed 7600 annealing steps and wins the race. Each consecutive shark has learned for 1500 fewer steps; the farthest shark has completed only 100 steps.

Figure 7.13 shows two frames from an animation of a similar scenario for the model of Kuhl's stingray (*Dasyatis kuhlii*). The ray closest to the viewer has completed 4560 annealing steps. Each consecutive ray has completed 900 fewer tries; the farthest ray having completed only 60 tries.

7.6 Abstraction of the Control Information

For both the fish and the snake we show the results of the parameterization for straight motion and for the left turn. Since the models are symmetric, right turn results can be obtained by exchanging the controls from the left side of the body with these from the right side. For the ray model we present only the results of the parameterization for straight motion. Each figure in this section shows, on top, learned control functions in the time domain. The center part of each figure illustrates the main modes of the Fourier transform for each of the control functions. The radius of the small circles depends on the amplitude of the mode, the position specifies the phase of the mode, and the distance from the center of the big circle is the frequency of the mode. The bottom part of the figure shows abstracted control functions using the modes of the FFT indicated above. The dotted lines illustrate controls for the left side of the model's body while the solid lines illustrate controls for the right side.

7.6.1 Ray

For the straight motion of the ray, it was sufficient to retain the dominant mode of the Fourier transform to obtain a good approximation of the functions. Figure 7.14 shows the learned controls for the ray swimming straight. Control functions for the elevator muscles are shifted 180 degrees with respect to the control functions for the depressor muscles.

7.6.2 Fish

For both the straight motion and the left turn of the fish, it was sufficient to retain the dominant mode of the Fourier transform to obtain good approximation of the functions. Figure 7.15A shows learned controls for the fish swimming straight. Control functions for the left side of the body are shifted 180 degrees with respect to the control functions for the right side. The back set of the muscles contracts with half the frequency rate of the other muscles and the muscles on each side of the body are activated in sequence with a slight shift in phase with respect to each other. For the swim left sequence (figure 7.15B) the back and front muscles on the right side of the body are essentially inactive, while all the muscles on the left side of the body move approximately in phase.

Figure 7.16 shows three frames from the animation that demonstrates results of locomotion control abstraction and the use of sensory input to accomplish a target following task. By using the high-level learning process described in section 7.6 of the thesis, the leopard shark has automatically abstracted six parameterizations to accomplish this task: do nothing, go straight, sharp turn left, sharp turn right, wide turn left, and wide turn right. The shark must locomote to successive targets indicated by the spheres. The black lines trace the successive positions of the target. Figure 7.18 shows the path that the shark follows as it tries to track the goal specified in the animation.

7.6.3 Snake

We had to use the two primary modes of the Fourier transform to get a sufficiently good abstraction for the turning snake. For the straight motion case (Figure 7.17A) the direct interpretation of the control functions is difficult, but if we look at the FFT representation clearly shows that the main modes have the same frequency and are slightly shifted in phase. This is what we would expect. When a snake wants to turn, it first coils itself in the direction of the turn then switches to its normal S like mode of locomotion. This sort of behavior is clearly seen in Figure 7.17B. First, all the muscles on the right side of the body expand and all the muscles on the left side contract. Then they resume the straight motion behavior.

7.7 Creation of Animations

When creating animations in from high level control abstractions, for each control window we search through the model's list of abstracted parameterizations to find the one that optimizes the objective function. When the search is finished, we use it to perform the forward simulation. We repeat this process for each control window, until the animation is finished. The algorithm performing this task can be written using a high level description language as:

```
while (animationNotFinished) {
  MIN = Infinity;
  for each parameterization(i) {
    compute N timesteps using parameterization(i);
    compute objectiveFunctionValue;
    if (objectiveFunctionValue < MIN) {
      newParameterization = parameterization(i);
      MIN = objectiveFunctionValue;
    }
    backup N timesteps;
  }
```

```
  compute N timesteps using newParameterization;
}
```

Figure 7.18 illustrates the application of control abstraction to a target following task. The model learned six parameterizations to accomplish this task: do nothing, go straight, sharp turn left, sharp turn right, wide turn left, and wide turn right. The model must locomote to successive targets indicated by black dots.

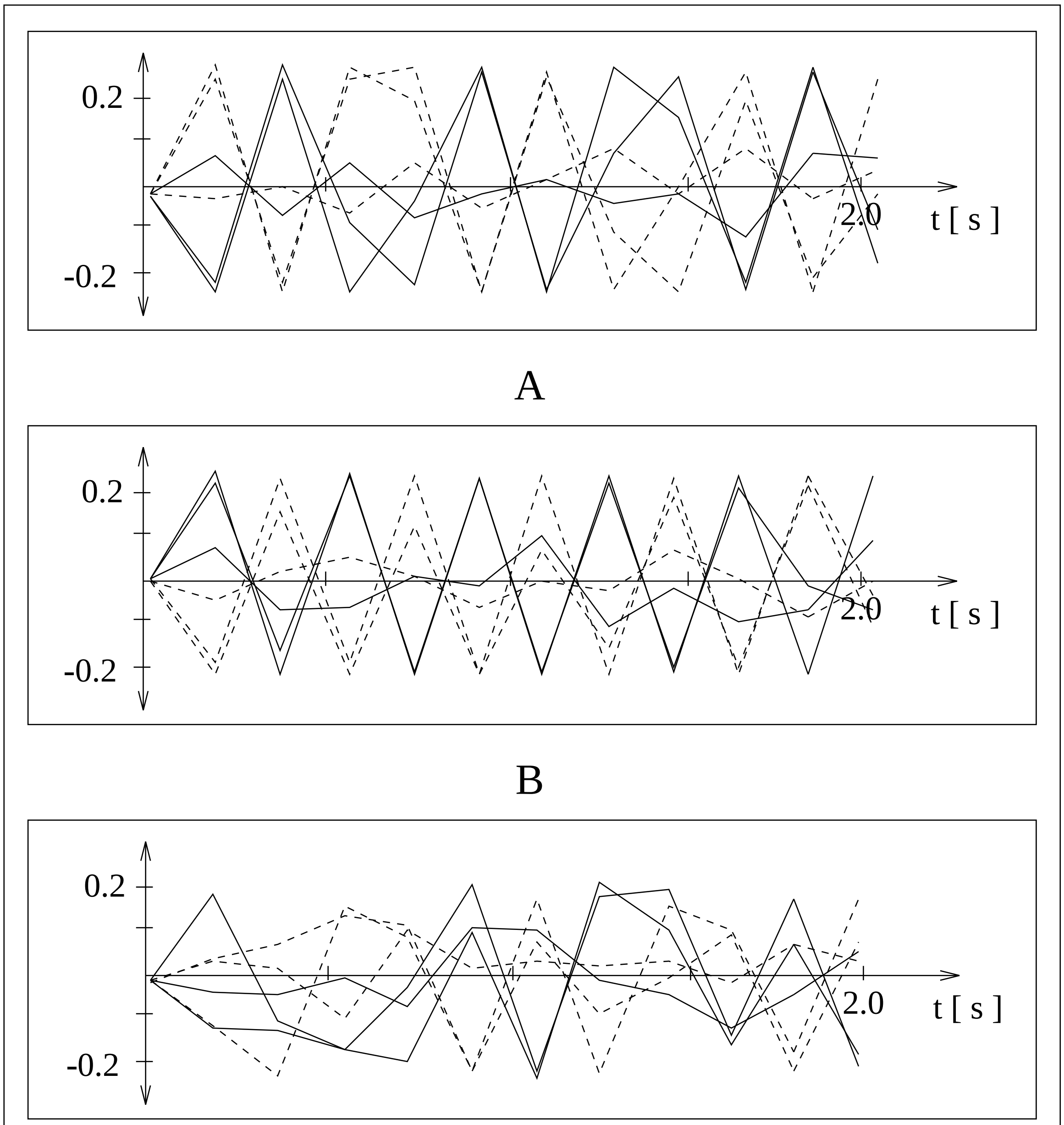

Figure 7.6: A shows the results obtained using the ascent algorithm for a single initial configuration. B shows the results obtained using the simulated annealing. C shows the results obtained using the random search.

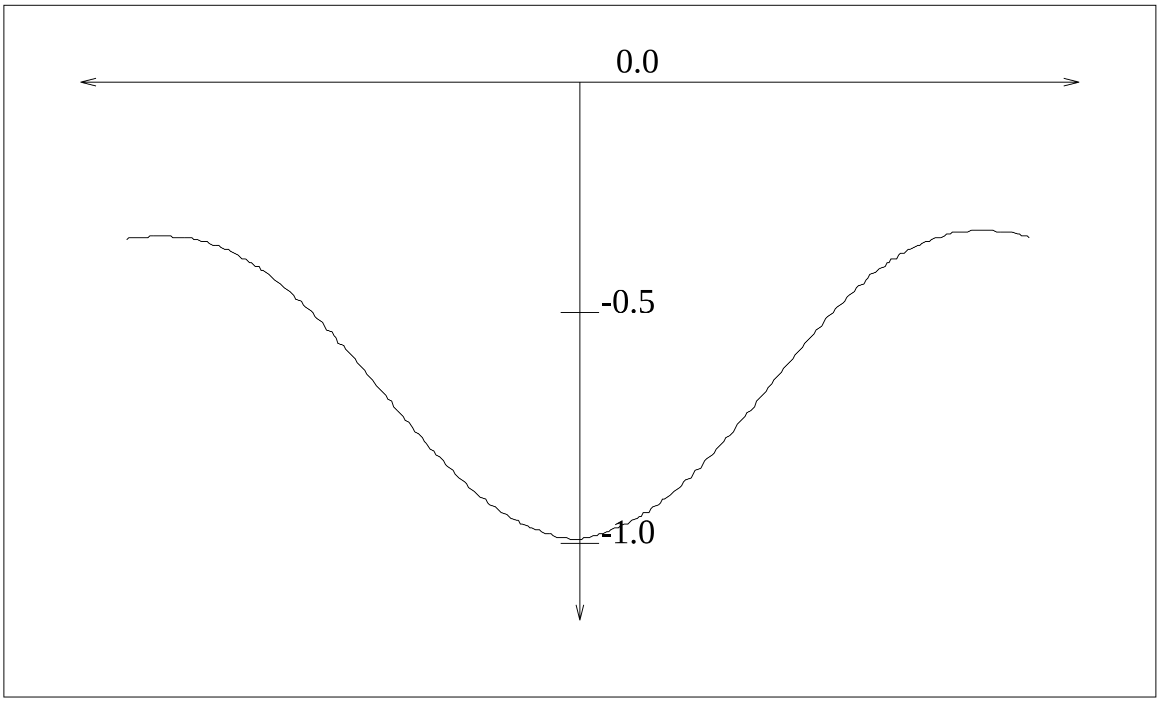

Figure 7.7: Moving along a diagonal line starting at the global maximum to the point that is symmetric with respect to the origin we obtain a symmetric function with a minimum centered at the origin and two global maxima at the boundaries of the line.

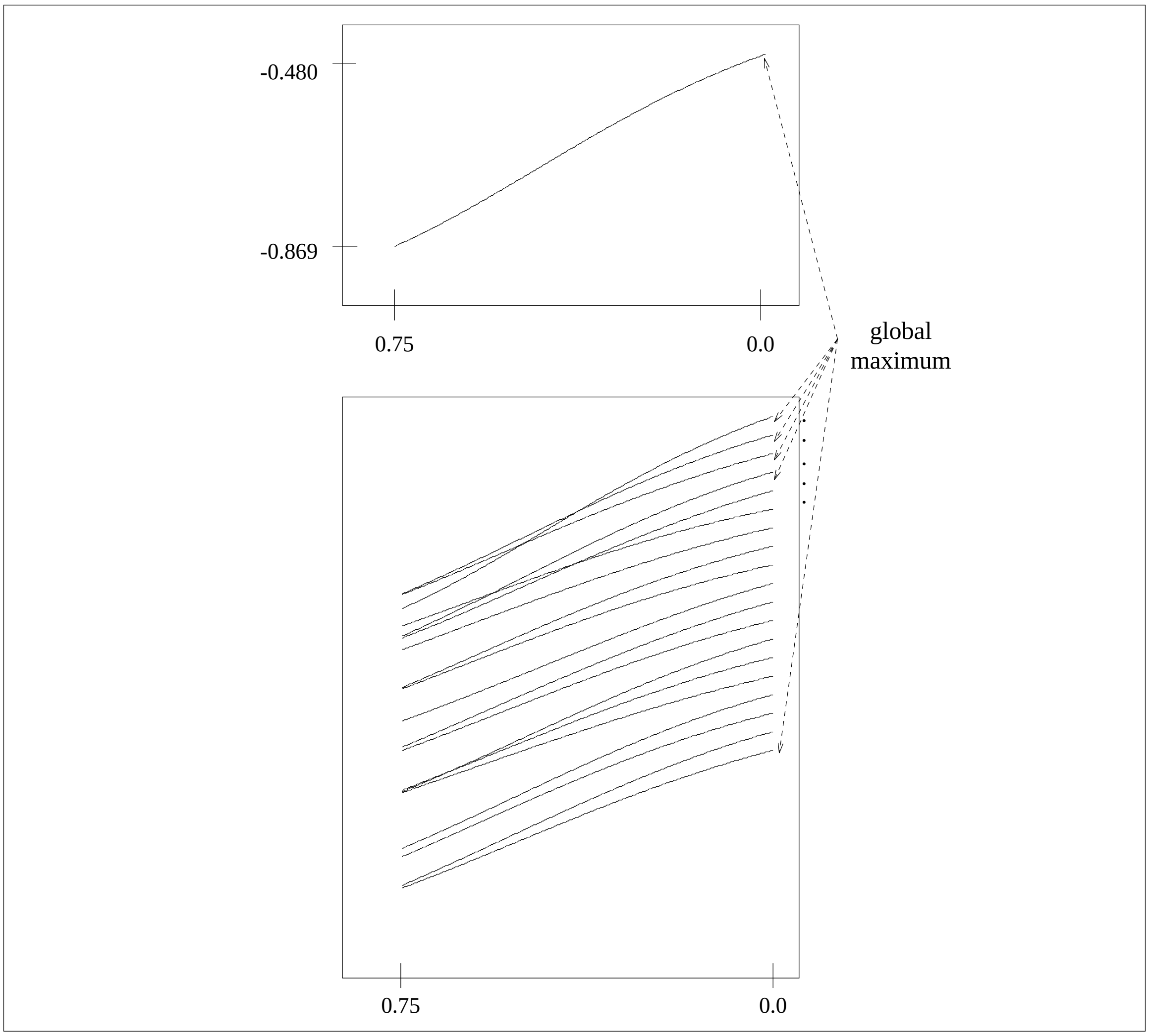

Figure 7.8: The objective function topography for the fish model. The top window indicates the values of the objective function at the starting and the final points of the line along which we evaluate the function. The bottom window shows the plots of the objective function along twenty different, randomly chosen lines. The numbers on the vertical axis indicate the Euclidean distance in the domain of the function from the point generating the global maximum.

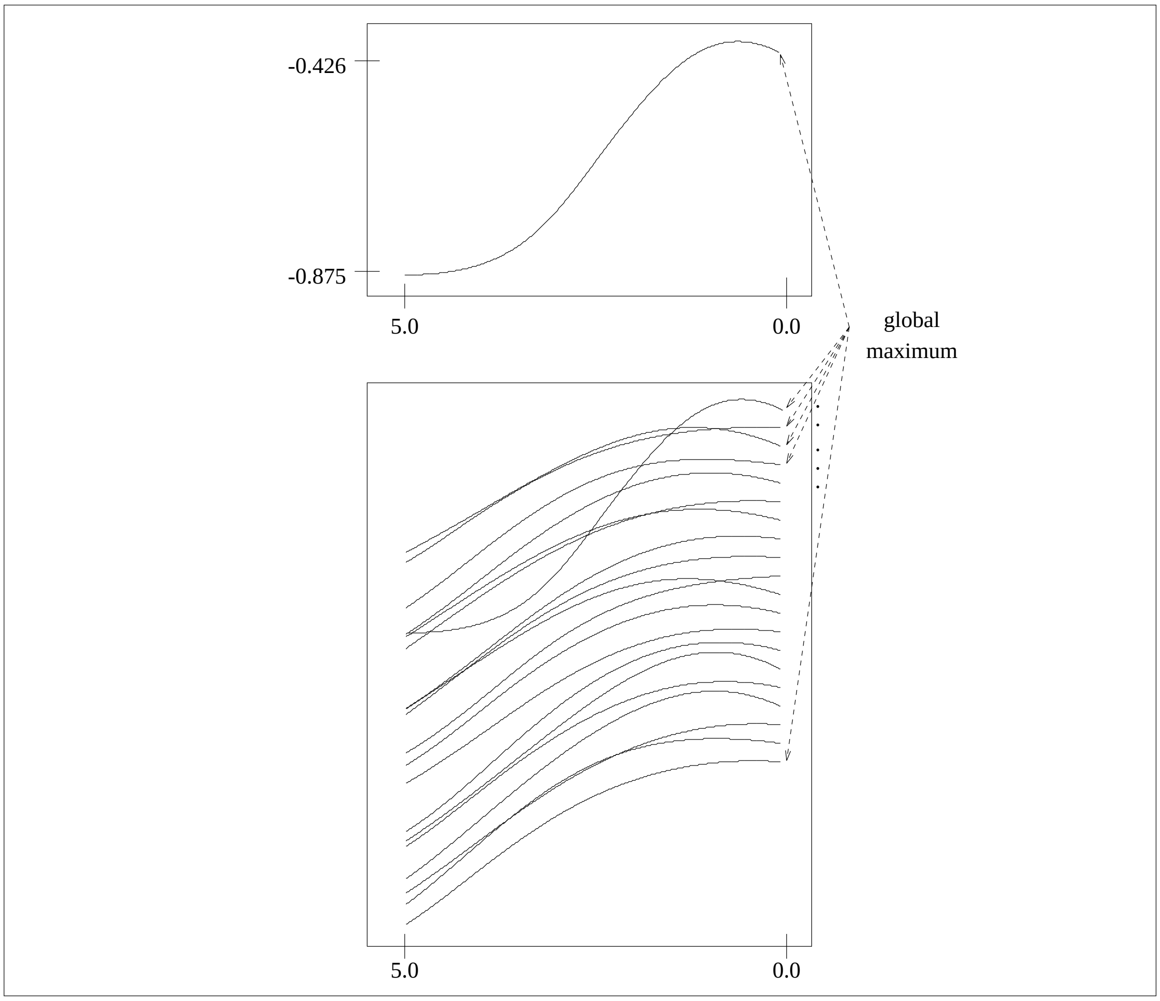

Figure 7.9: The 0bjective function topography for the snake model. The top window indicates the values of the objective function at the starting and the final points of the line along which we evaluate the function. The bottom window shows the plots of the objective function along twenty different, randomly chosen lines. The numbers on the vertical axis indicate the Euclidean distance in the domain of the function from the point generating the global maximum.

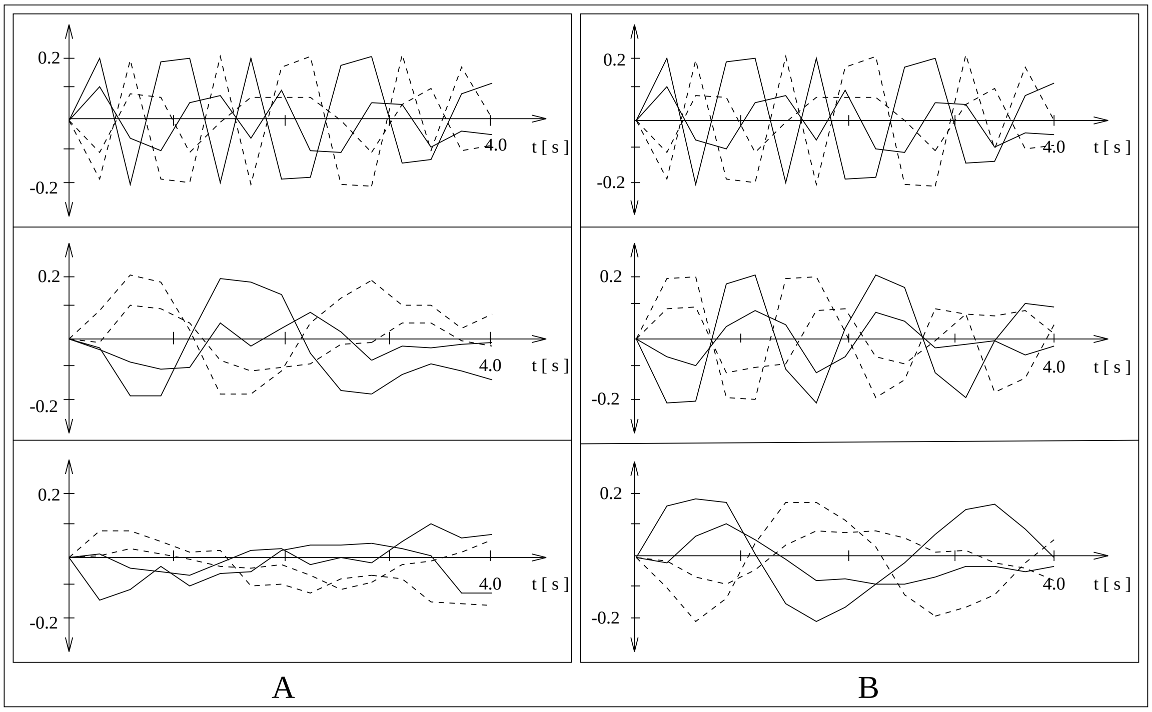

Figure 7.10: The influence of the E_{c1} (A) and E_{c2} on the control functions

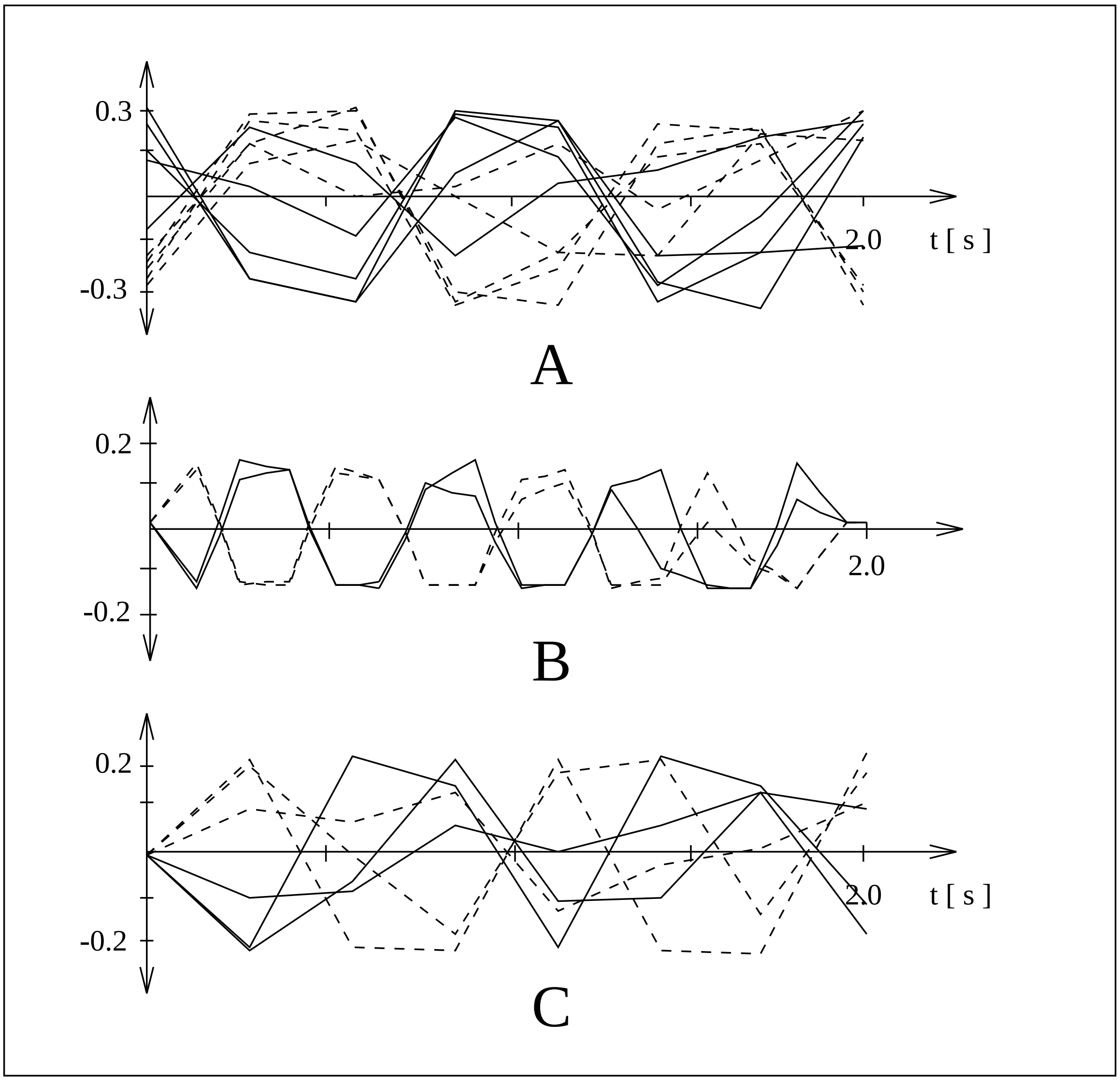

Figure 7.11: The results of low level control learning for three different models: snake(A), ray(B), fish(C).

Figure 7.12: Two frames from the animation showing the learning process of leopard shark *Triakis semifasciata*.

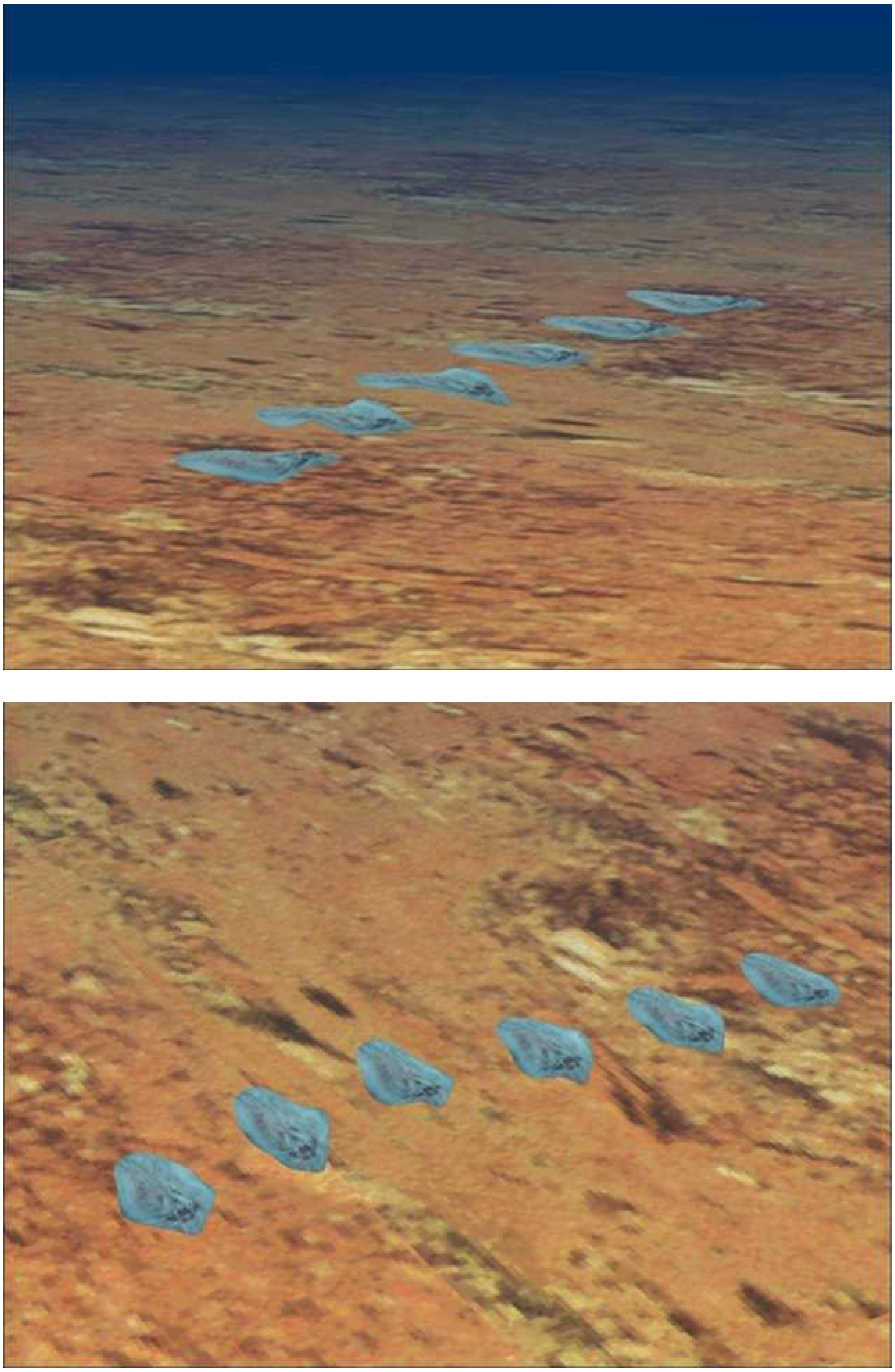

Figure 7.13: Two frames from the animation showing the learning process of Kuhl's stingray *Dasyatis kuhlii*.

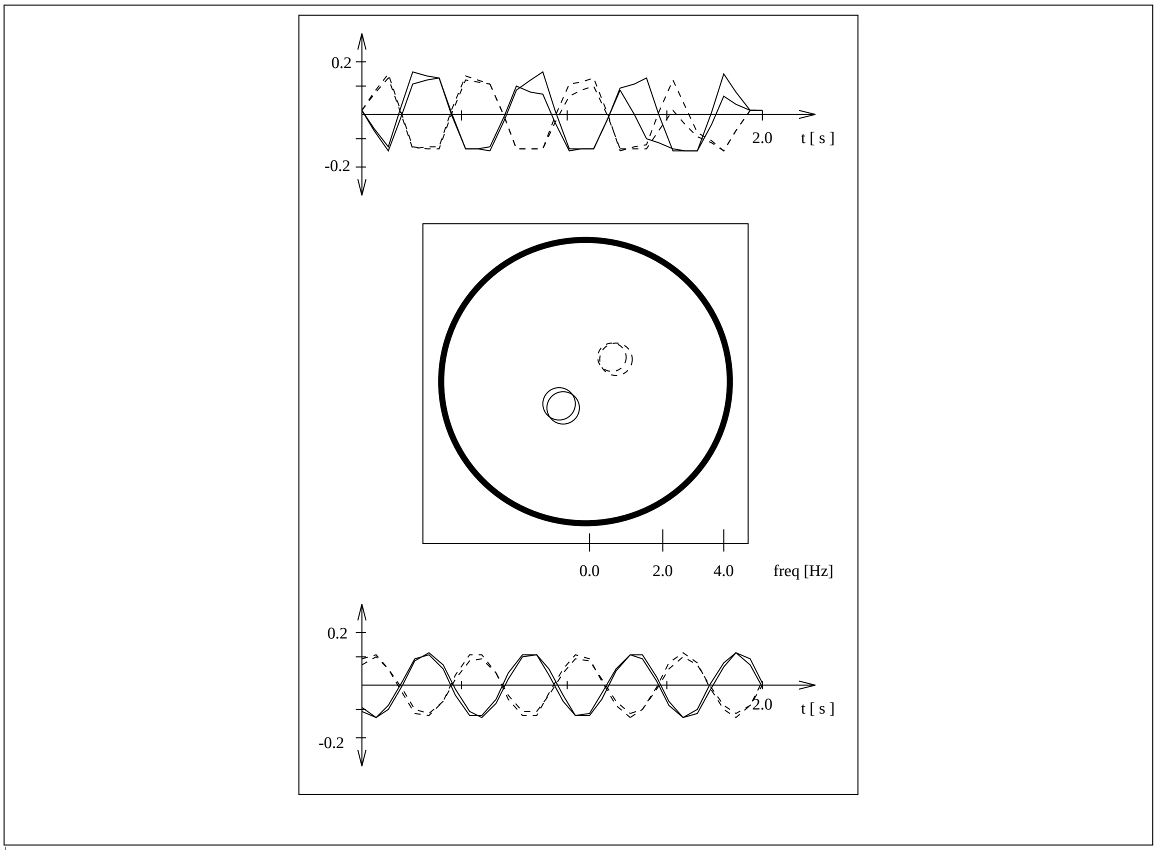

Figure 7.14: The results for the straight swim for the ray

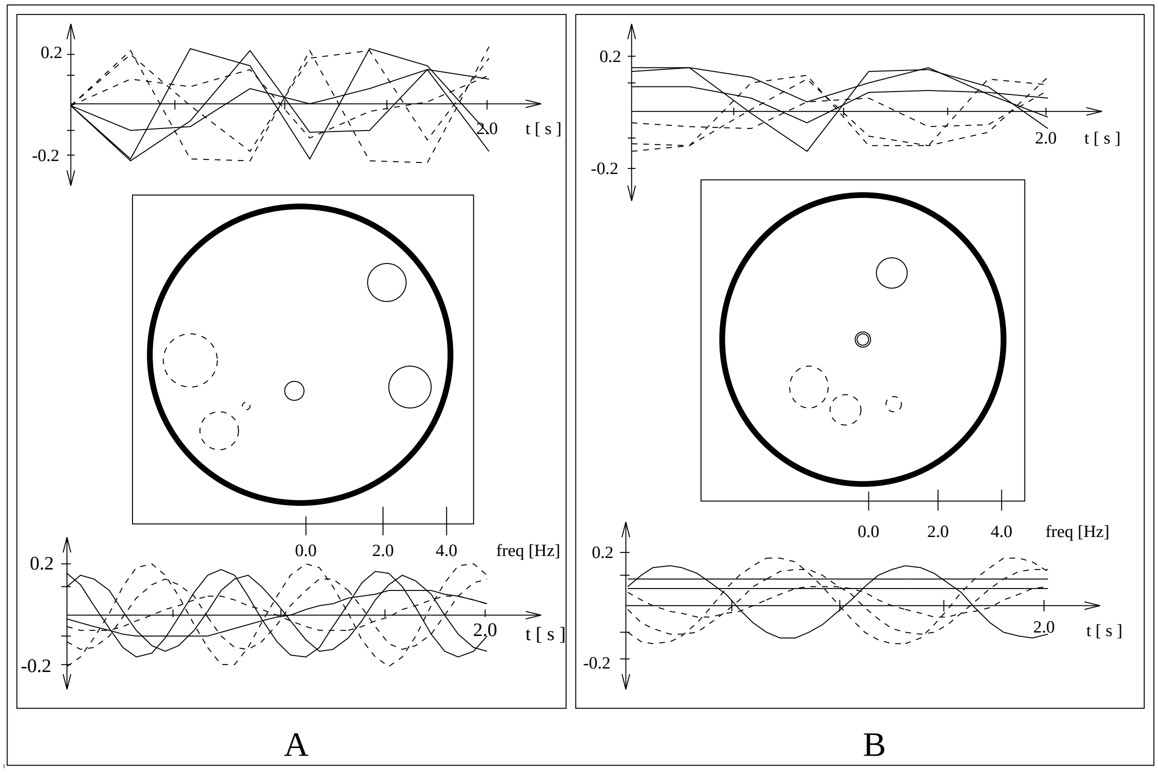

Figure 7.15: The results for the straight swim and the left turn for the fish

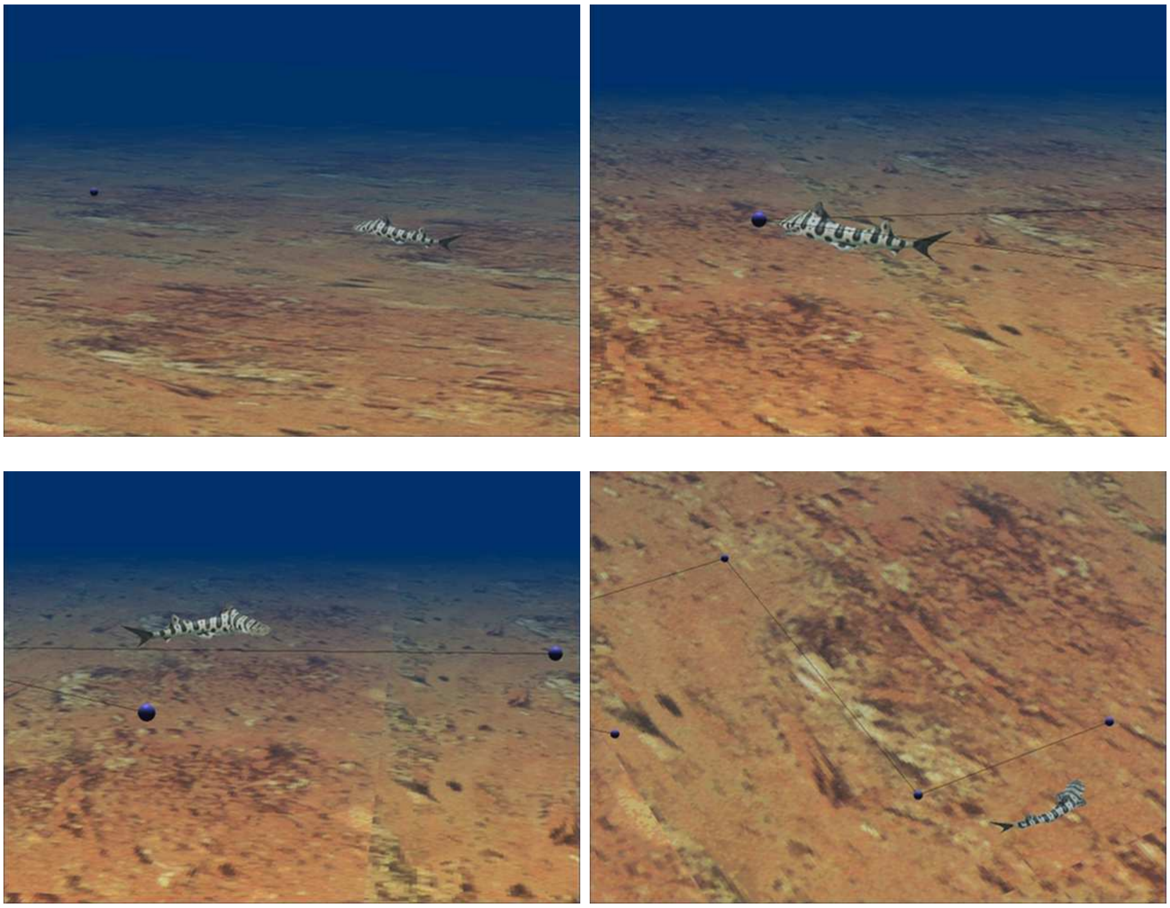

Figure 7.16: Four frames from the animation showing the results of locomotion control abstraction and the use of sensory input to accomplish a target following task. The shark must locomote to successive targets indicated by the ball. The black lines trace the successive positions of the target.

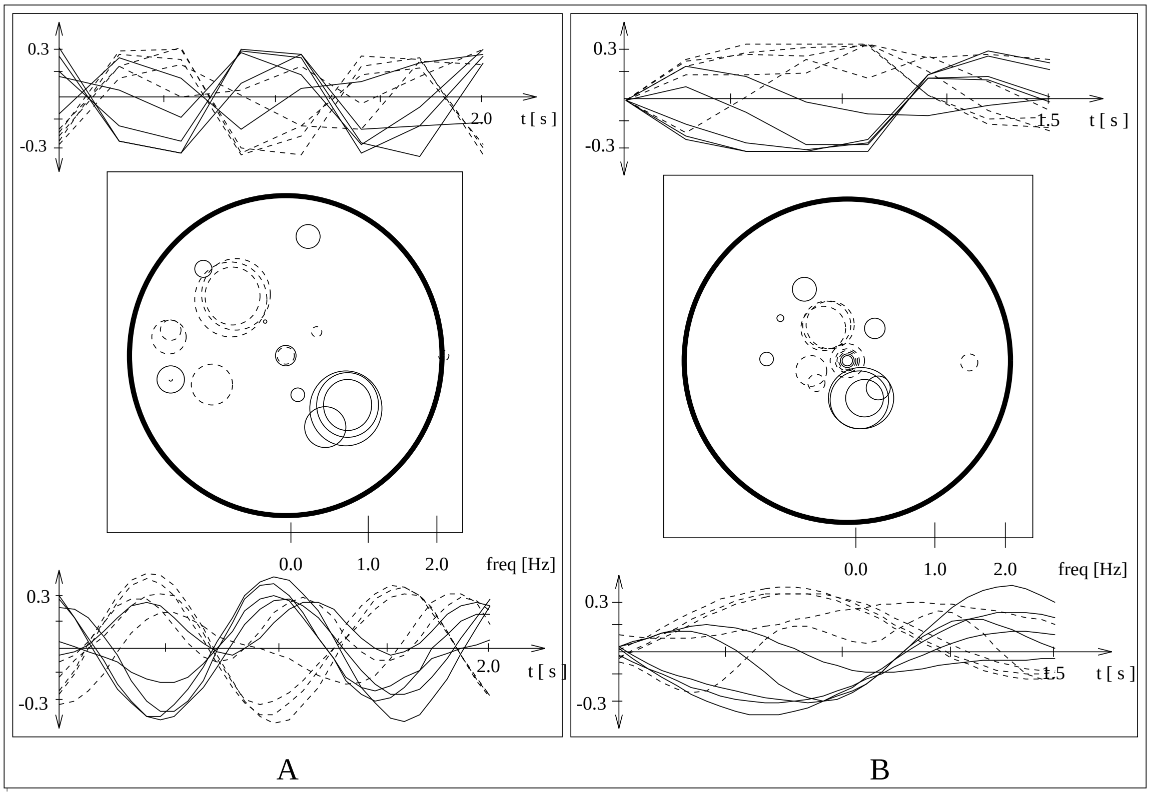

Figure 7.17: The results for the straight motion and the left turn for the snake

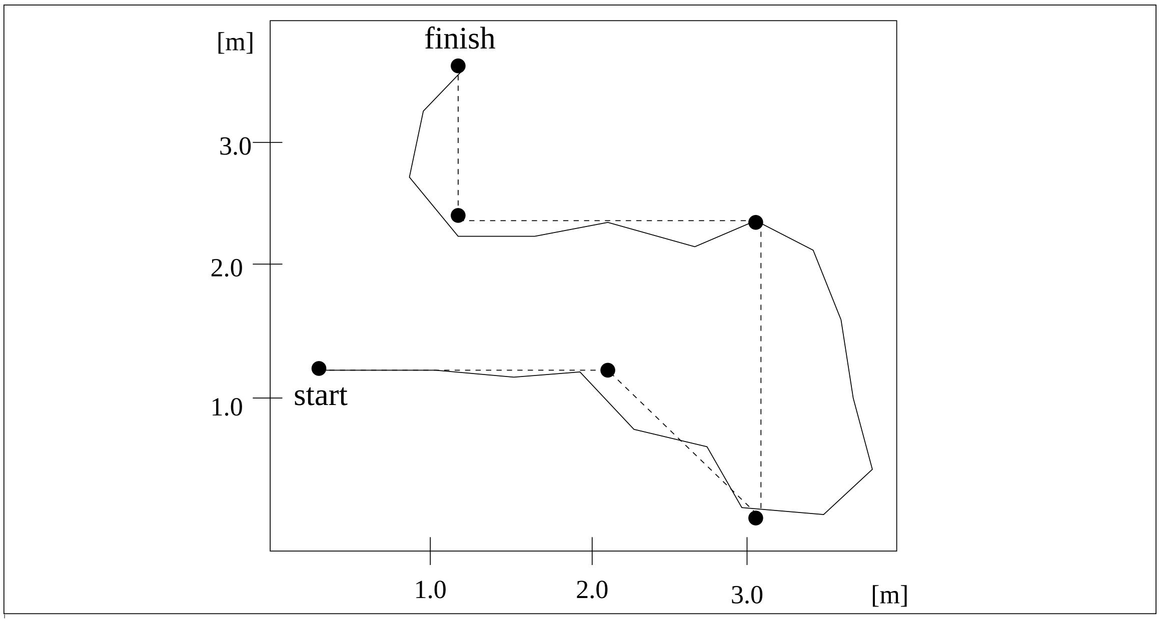

Figure 7.18: Goal Tracking. Black dots indicate consecutive positions of the goal. The solid line indicates the path of the shark tracking the goal.

Chapter 8

Conclusion

8.1 Summary

We have presented a new method for the learning and emulation of natural locomotion patterns. We reduce the dimensionality of the optimization process by performing the search in the actuator space of the models, instead of in the state space. A global optimization technique is used to obtain realistic results. To make the approach practical we divide the learning process into two separate stages. The first stage finds a solution to the low level control problem and the second stage abstracts the control information for future use. The models are equipped with sensors that enable dynamic response to changes in the environment. To be able to influence the motion, we introduce into the objective function a term evaluating the muscle activation functions. In this way we can control velocity, energy expenditure, and the smoothness of the muscle activation functions. By enabling our models to learn autonomously how to control themselves, we free the animator from the necessity of having to know the physical structure of the animate model in order to achieve realistic animation.

8.2 Discussion

The lack of a keyframe interpolation term in our objective function can limit significantly its applicability to animation production. Suppose, for example, that we want to animate the swing of a simple pendulum, and we want the pendulum to stop at the apex of its swing after completing two full swings, as illustrated in Figure 8.2. Without incorporating some form of keyframe interpolation term into the objective function, we will not be able to produce this very simple animation. In fact, as the large volume of work from numerous animation production houses clearly demonstrates, keyframe interpolation is oftentimes sufficient to create spectacular animations. That is why keyframe interpolation is likely to remain an integral part of production animation systems in the foreseeable future.

Our work has not attempted to supplant the keyframe animation technique in general. However, our goal has been to develop a different approach that may be more suitable for a certain class of applications—the realistic animation of complex environments populated by autonomous creatures that can locomote thorough internal muscle actions in accordance with physical principles. Automatically learning this form of locomotion is important not only for its potential application to computer animation, but also by what it can may potentially reveal about similar learning processes that occur naturally, although we have not been concerned with the latter issue in this thesis.

8.3 Research Directions

Adding a keyframe interpolation term into the objective function would dramatically increase the usefulness of our technique for animation production. Such an extension would probably require the use of constraint methods. For example, penalty constraints can be easily applied to closely approximate keyframes that specify the complete configuration of the model. A penalty term can be added to the objective function that minimizes the negative squared distance of the configuration of the

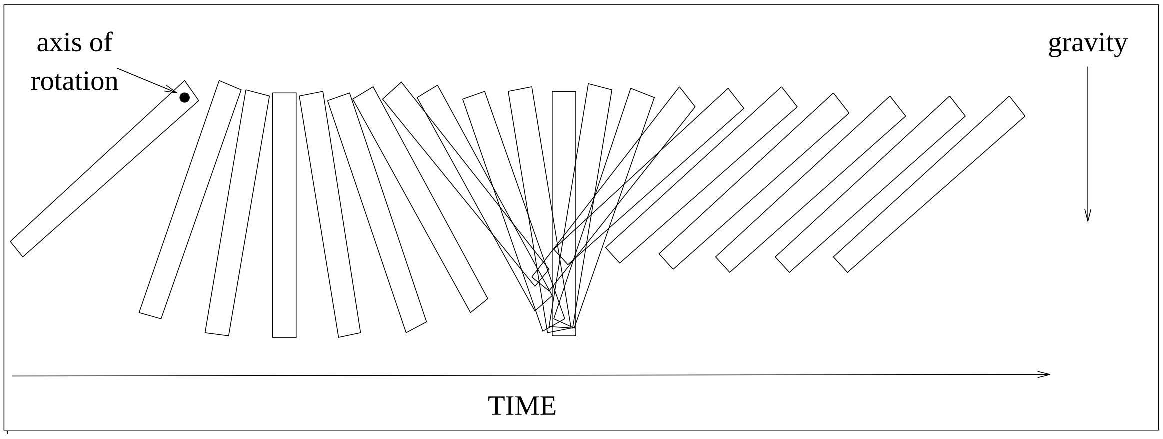

Figure 8.1: The figure shows a number of keyframes from the animation of a pendulum swing. The pendulum stops at the apex of its swing after completing two full swings by exerting internal control at the axis of rotation.

model at time t_k from the given keyframe configuration at that time for some distance metric (Figure 8.3). Typically, the animator would not wish to specify the complete configuration for a complex animate model, but, rather, derived quantities such as the center of mass of the model. Such derived quantities can be keyframed through similar constraints.

Our animate models can obviously be improved. The ray model is not flexible enough to be able to achieve reasonable speed of locomotion and sufficient maneuverability. Figure 8.3 shows the top view of half of an improved ray model. Equipping the fish model with pectoral fins (as in [37]), in addition to the caudal fin that it now has, would improve its finesse and permit vertical motion. Learning to control a more complete set of fins presents an interesting problem for future research. We intend to conduct more research into the biology literature on mechanisms for coordinating complex patterns of animal movement.

Our recent experiments lead us to believe that the objective function in certain cases might be not as densely populated with local suboptima as we originally thought. It might in fact can be quite well behaved. It is possible that in certain cases iterative optimization methods based on numerically estimated gradients might converge to a good solution much quicker than simulated annealing. To answer that question, it will be helpful to investigate the topography of the search space more systematically

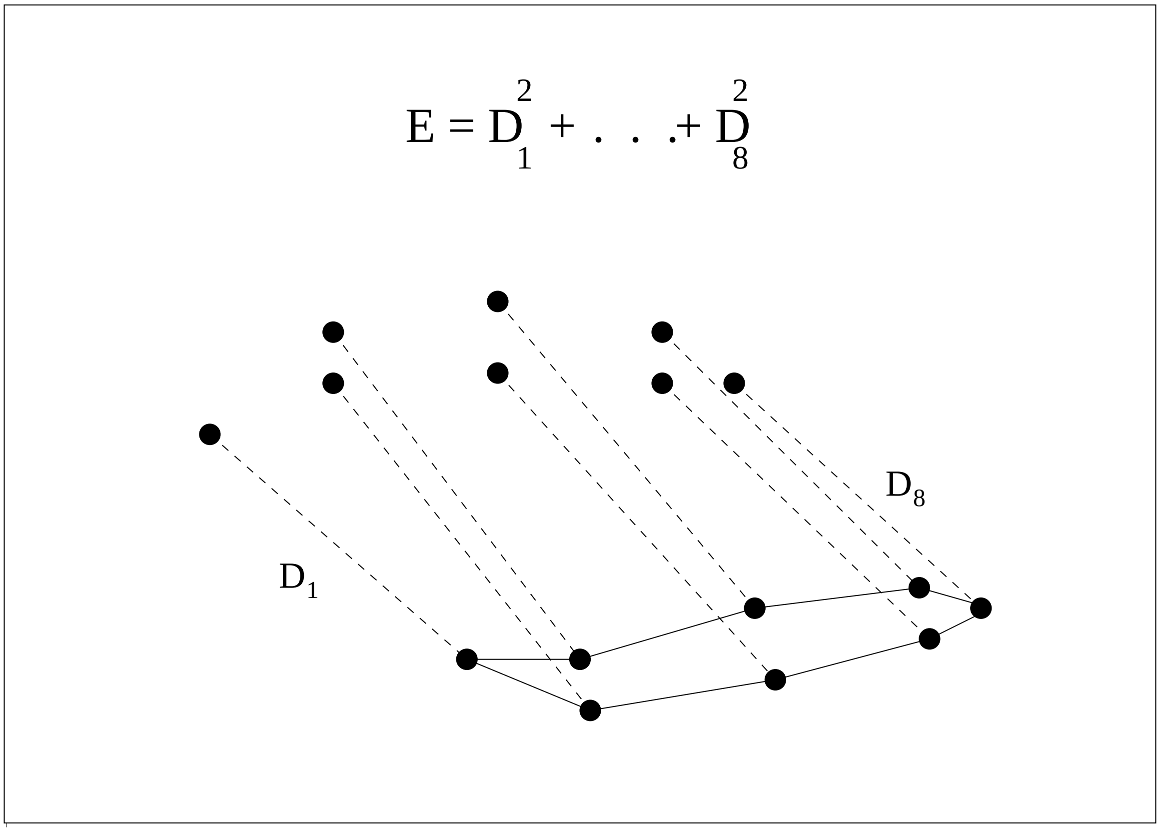

Figure 8.2: A possible way of achieving keyframe interpolation. The 8 markers specify the desired configuration of the models and 8 corresponding markers on the fish's body. The evaluation metric is the sum of the Euclidean distances between the pairs of markers.

for a greater variety of objective functions.

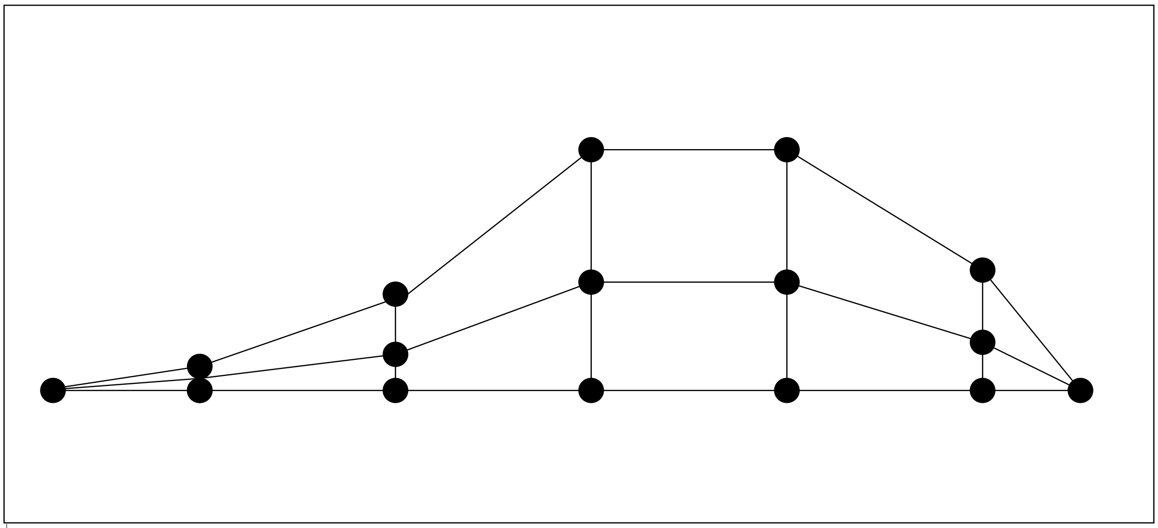

Figure 8.3: Improved ray model. The black dots indicate point masses. The figure shows the top view of the left half of the model.

Appendix A

Mass-Spring Models

The mechanics of the spring-mass model are specified as follows: Let node i have mass m_i, position $\mathbf{x}_i(t) = [x_i(t), y_i(t), z_i(t)]$, velocity $\mathbf{v}_i(t) = d\mathbf{x}_i/dt$, and acceleration $\mathbf{a}_i(t) = d^2\mathbf{x}_i/dt^2$. Let spring S_{ij} connect node i to node j. It has spring constant c_{ij} and natural, rest length l_{ij}. Its deformation is $e_{ij}(t) = ||\mathbf{r}_{ij}|| - l_{ij}$, where $\mathbf{r}_{ij} = \mathbf{x}_j(t) - \mathbf{x}_i(t)$. The force S_{ij} exerts on node i is $\mathbf{f}^s_{ij} = c_{ij}e_{ij}(t)\mathbf{r}_{ij}/||\mathbf{r}_{ij}||$ (and it exerts the force $-\mathbf{f}^s_{ij}$ on node j). The Lagrange equations of motion are:

$$m_i\ddot{\mathbf{x}}_i(t) + \gamma_i\dot{\mathbf{x}}_i(t) - \mathbf{f}_i^{inter}(t) - \mathbf{f}_i^{exter}(t) = 0 \tag{A.1}$$

where γ_i is the damping factor, $\mathbf{f}_i^{inter}(t) = \sum_{j\in N_i} \mathbf{f}^s_{ij}(t)$ is the net internal force on node i due to springs connecting it to nodes $j \in N_i$, where N_i is the index set of neighboring nodes. Finally, $\mathbf{f}_i^{exter}$ is the external force on node i.

A.1 Forward Simulation

Ideally, we would like to use an integration method that quickly gives us an accurate and stable solution. Unfortunately, it is rather hard to find a method satisfying all these criteria simultaneously. More accurate methods are also more costly. The explicit Euler method is the fastest. However, it is also completely impractical in our application, because it is of limited accuracy and because it becomes unstable quite easily. If we want to obtain meaningful results using this method, we need to use a very small time step, but this hurts the performance. To advance a solution from t_i to $t_{i+1} = t_i + h$, this method uses derivative information only from the previous step. This approach leads to the big error term of $O(h^2)$ (see Figure A.1). Even more important in our case, is the explicit method's poor performance on stiff equations. Our models quite often require very stiff springs leading to stiff equations of motion for which this method is useless.

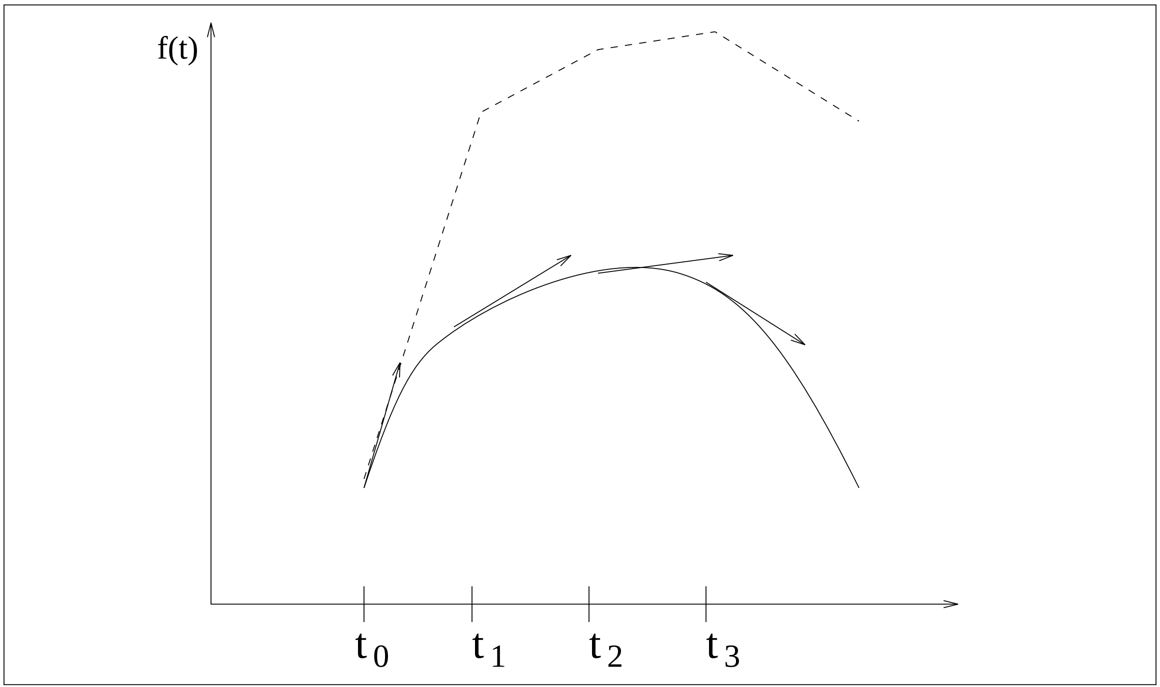

Figure A.1: Explicit Euler Method. Using derivative information only at the beginning of the interval results in bad accuracy. The solid line shows the function being approximated. The dotted line shows the approximation. The arrows indicate the direction of the derivative at a particular time step.

The implicit integration procedures are more stable, but they are also much more expensive. To make the derivative information more accurate, we first take a trial step from t_i to t_{i+1}, compute the derivatives for t_{i+1}, and use this information for making the actual step. This results in a higher order integration methods that are usually more accurate. Figure A.2 illustrates an implicit integration method, where the trapezoidal rule [16] was used to approximate the derivative information. This approach uses the average of the derivative information at both t_i and t_{i+1}.

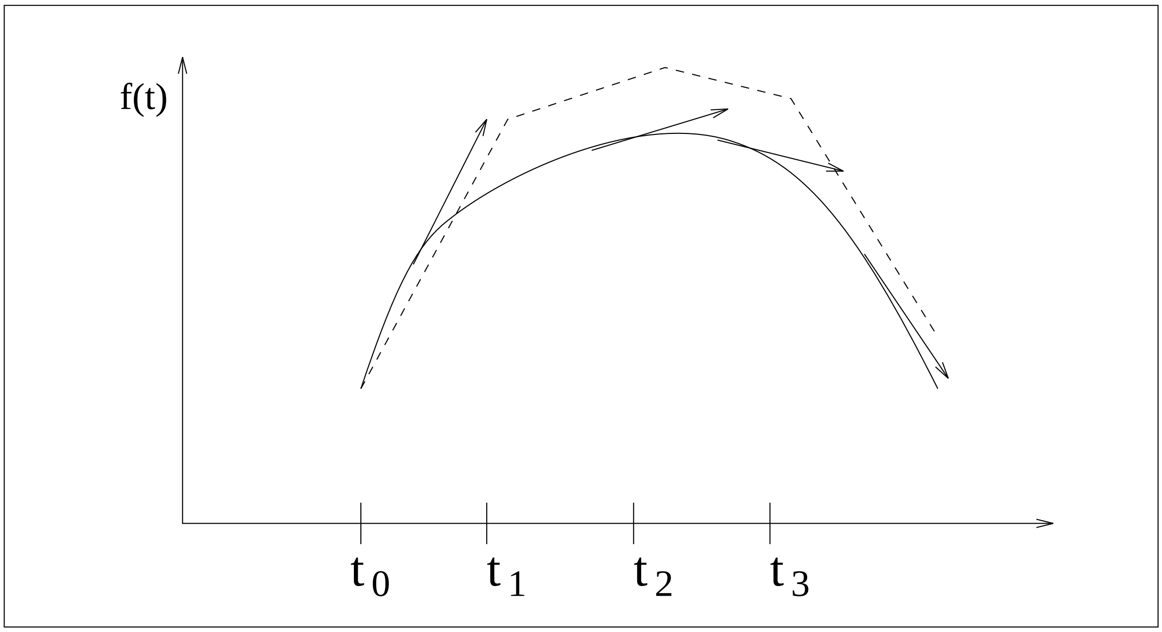

Figure A.2: Implicit Integration Method. Taking the trial step to better approximate the derivative, leads to much more accurate results. The solid line shows the function being approximated. The dotted line shows the approximation. The arrows indicate the direction of the derivative at a particular time step.

For the integration scheme we used the semi-implicit Euler method [31]that is a combination of both explicit and implicit methods. It computes $\mathbf{f}_{i+1}^{exter}$ explicitly and $\mathbf{f}_{i+1}^{inter}$ implicitly. This leads to a fairly stable solution that is also relatively cheap to evaluate.

A.2 Matrix Representation

Let $\mathbf{K}$ denote the stiffness matrix. It is a square matrix of size N, where N is the number of particles in the model. If $k_{ij}(t)$ is the stiffness of the spring between the nodes i and j, then $\mathbf{K}_{ij} = k_{ij}(t)$, $\mathbf{K}_{ii} = \mathbf{K}_{jj} = -2k_{ij}(t)$ if the spring between nodes i and j exists, and $\mathbf{K}_{ij} = 0.0$ otherwise. Let $\mathbf{C}$ denote a diagonal matrix of size N, where $\mathbf{C}_{ii} = \frac{m_i}{h}$ and, let $\mathbf{\Gamma}$ be a diagonal matrix of size N, such that $\mathbf{\Gamma}_{ii} = \gamma_i$. Using this notation the Lagrangian equation of motion A.1 can be represented as

$$\begin{aligned}
\mathbf{A}_{i+1} &= -\mathbf{K}_i h + \mathbf{C} \\
\mathbf{b}_{i+1} &= (\mathbf{C} - \mathbf{\Gamma})\mathbf{v}_i + \mathbf{K}_i \mathbf{x}_i + \mathbf{f}_i^{exter} \\
\mathbf{v}_{i+1} &= \mathbf{A}_{i+1}^{-1} \cdot \mathbf{b}_{i+1} \\
\mathbf{x}_{i+1} &= \mathbf{x_i} + \mathbf{v}_{i+1} h.
\end{aligned}$$

Most of the matrices used here are sparse. This is particularly evident when we are simulating models with a large number of nodes and a small number of springs. In a situation like that, it is good to take an advantage of a special storage scheme to improve performance and decrease the space requirement. However, in practical applications we usually deal with small models with a significant number of springs, for which the matrices are not very sparse and not very large.

A.3 The LU Decomposition

Forward simulation step using the matrix representation can be formulated in a very compact, clear way.

```
void
simulationStep(void) {
    calculateExternalForces();
    construct_K();
    construct_A();
    construct_b();
    updateVelocities();
    updatePositions();
}
```

In order to compute the velocities, we need to solve the matrix equation

$$\mathbf{A}_{i+1} \cdot \mathbf{v}_{i+1} = \mathbf{b}_{i+1}.$$

We use the LU decomposition to do this [31]. This method decomposes the matrix $\mathbf{A}_{i+1}$ into a product of two matrices

$$\mathbf{A}_{i+1} = \mathbf{L}_{i+1} \cdot \mathbf{U}_{i+1}$$

where $\mathbf{L}_{i+1}$ is the lower triangular and $\mathbf{L}_{i+1}$ is the upper triangular. Instead of solving the original equation, we solve first the equation $\mathbf{L}_{i+1} \cdot \mathbf{y}_{i+1} = \mathbf{b}_{i+1}$ by forward substitution and then $\mathbf{U}_{i+1} \cdot \mathbf{v}_{i+1} = \mathbf{y}_{i+1}$ by backward substitution. This is a useful scheme, since once we have the LU decomposition of $\mathbf{A}_{i+1}$, we can solve for many right hand sides very cheaply. In our case $\mathbf{v} = [\mathbf{v}_x, \mathbf{v}_y, \mathbf{v}_z]$ and we need 3 right hand side evaluations to obtain the solution.

Appendix B

Model Description using a Script Based Language

To simplify the process of building the creatures easier, we wrote a simple interpreter that reads all the necessary information about the model from a script file. The information includes the initial positions of the particles and their masses, indices of the end points for all the springs and triples of indices that make up the triangular faces of the creatures body.

Figure B.1 shows the simplest possible 3D model for which we provided the script definition. It is a tetrahedron equipped with 3 muscles.

```
#script for a tetrahedron with 3 muscles

4                # the number of particles

# index    position       mass

    1    [ 0.0 0.0  0.0 ]   1.0
    2    [ 1.0 0.0  0.0 ]   1.0
    3    [ 1.0 1.0  1.0 ]   1.0
```

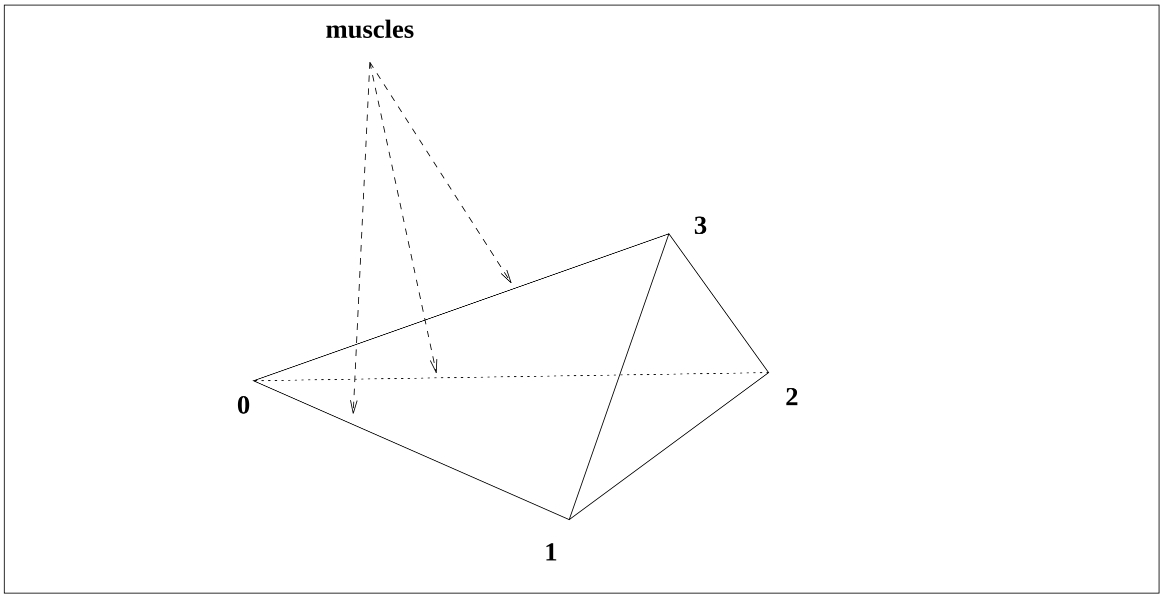

Figure B.1: A tetrahedron equipped with 3 muscles.

```
    4   [ 1.0 1.0 -1.0 ]   1.0

6                 # the number of springs

# index start  end  stiffness  active

    1     0     1     0.2        1     # active == 1 indicates that
    2     0     2     0.2        1     # the spring is capable of
    3     0     3     0.2        1     # exerting control
    4     1     2     0.2        0
    5     2     3     0.2        0
    6     1     3     0.2        0

4                 # the number of faces

# index   A   B   C

    1     0   1   3      # make sure that the normals
```

```
2     0  1  2      # are pointing in the right direction
3     0  2  3
4     1  2  3
```

Bibliography

[1] J.H. Ahlberg, E.N. Nilson, and J.L. Walsh. *The Theory of Splines and their Applications*. Academic Press, New York, 1967.

[2] W. W. Armstrong and W. Green. The dynamics of articulated rigid bodies for purpose of animation. *The Visual Computer*, 19(4):23, 1985.

[3] N. I. Badler, K. H. Manoochahri, and Walters G. Articulate figure positioning by multiple constraints. *IEEE Trans. Computer Graphics and Animation*, page 28, 1987.

[4] D. Baraff and A Witkin. Dynamic simulation of non-penetrating flexible bodies. Proceedings of ACM SIGGRAPH, 26(2):303–311, 1992.

[5] R. Barzel and A. H. Barr. A modeling system based on dynamic constraints. Proceedings of ACM SIGGRAPH, 22(4):179–187, 1988.

[6] R. A. Brooks. A robot that walks: Emergent behaviours from a carefully evolved network. In *Making Them Move*, pages 99–108. Morgan Kaufmann, San Mateo, California, 1991.

[7] L. S. Brotman and A. N. Netravali. Motion interpolation by optimal control. Proceedings of ACM SIGGRAPH, 22(4):309–407, 1988.

[8] A. Bruderlin and T. Calvert. Goal directed, dynamic animation of human walking. Proceedings of ACM SIGGRAPH, 23(3):233–241, 1989.

[9] J. I. Castro. *The Sharks of North American Waters*. Texas Unviersity Press, 1983.

[10] M. F. Cohen. Interactive spacetime control for animation. Proceedings of ACM SIGGRAPH, 26(2):293–301, 1992.

[11] C. H. Ernst. *Venomous Reptiles of North America*. Smithonian Institution Press, 1992.

[12] M. Girard. Constrained optimization of articulated animal movement in computer animation. In *Making Them Move*, pages 209–232. Morgan Kaufmann, San Mateo, California, 1991.

[13] M. Girard and A. A. Maciejewski. Computational modelling for the computer animation of legged figures. Proceedings of ACM SIGGRAPH, 19(2):263–271, 1985.

[14] P. M. Isaacs and M. F. Cohen. Controlling dynamic symulation with kinematic constraints, behaviour functions and inverse dynamics. Proceedings of ACM SIGGRAPH, 21(4):215–223, 1987.

[15] P. M. Isaacs and M. F. Cohen. Mixed methods for complex kinematic constraints in dynamic figure animation. *The Visual Computer*, 4:296, 1988.

[16] D. Kahaner, C. Moler, and S. Nash. *Numerical Methods and Software*. Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[17] D. H. Kochanek and R. H. Bartels. Interpolating splines for keyframe animation. *Graphics Interface*, page 41, 1984.

[18] P. J. M. van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, Dordrecht, Holland, 1987.

[19] J. Lasseter. Principles of traditional animation applied to 3d computer amination. Proceedings of ACM SIGGRAPH, 21(4):35–43, 1987.

[20] M. McKenna and D. Zeltzer. Dynamic simulation of autonomous legged locomotion. Proceedings of ACM SIGGRAPH, 24(4):29–38, 1990.

[21] R. McNeill Alexander. *Exploring Biomechanics: Animals in Motion*. Scientific American Library, New York, 1992.

[22] D. Metaxas and D. Terzopoulos. Dynamic deformations of solid primitives with costraints. Proceedings of ACM SIGGRAPH, 26(2):309–317, 1992.

[23] G.S.P. Miller. The motion dynamics of snakes and worms. Proceedings of ACM SIGGRAPH, 22(4):169–177, 1988.

[24] J. T. Ngo and J. Marks. Spacetime constraints revisited. Proceedings of ACM SIGGRAPH, 27(2):343–351, 1993.

[25] M. van de Panne and E. Fiume. Sensor-actuator networks. Proceedings of ACM SIGGRAPH, 27(2):335–343, 1993.

[26] M. van de Panne, E. Fiume, and Z. Vranesic. Reusable motion synthesis using state-space controllers. Proceedings of ACM SIGGRAPH, 24(4):225–233, 1990.

[27] J. Park, D. Fussell, M. Pandy, and J. C. Browne. Realistic animation using musculotendon skeletal dynamics and supoptimal control. *Third Eurographics Workshop on Animation and Simulation*, September 1992.

[28] K. G. Pearson. Sensory elements in pattern-generating networks. In *Making Them Move*, pages 111–127. Morgan Kaufmann, San Mateo, California, 1991.

[29] C.B. Phillips and N.I. Badler. Interactive behaviours for bipedal articulated figures. Proceedings of ACM SIGGRAPH, 25(4):359–367, 1991.

[30] J. Platt and A. H. Barr. Constraint methods for flexible models. Proceedings of ACM SIGGRAPH, 22(4):279–287, 1988.

[31] W. H. Press, B. Flannery, et al. *Numerical Recipies: The Art of Scientific Computing*. Cambridge University Press, 1986.

[32] M.H. Raibert and J.K. Hodgins. Animation of dynamic legged locomotion. Proceedings of ACM SIGGRAPH, 25(4):349–357, 1991.

[33] J. E. Rendal, Allen G. R., and Steene R. C. *Fishes of the Great Barrier Reef and Coral Sea*. Unviersity of Hawaii Press, Honolulu, Hawaii, 1990.

[34] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4(6):306–331, 1988.

[35] D. Terzopoulos, J. Platt, et al. Elastically deformable models. Proceedings of ACM SIGGRAPH, 27(4):205–214, 1987.

[36] D. Terzopoulos, A. Witkin, and M. Kass. Constraints on deformable models, recovering 3d shape and nonrigid motion. *Artificial Intelligence*, 36:417–430, 1988.

[37] X. Tu, D. Terzopoulos, and E. Fiume. Go fish! ACM SIGGRAPH Video Review Issue 91: SIGGRAPH'93 Electronic Theater, 1993.

[38] J. Wilhelms and B. Barsky. Using dynamic analysis to animate articulated bodies such as humans and robots. *Graphics Interface*, page 97, 1985.

[39] A. Witkin, K. Fleicher, and A. Barr. Energy constraints on parametrized models. Proceedings of ACM SIGGRAPH, 21(4):225–233, 1987.

[40] A. Witkin and M. Kass. Spacetime constraints. Proceedings of ACM SIGGRAPH, 22(4):159–167, 1988.

[41] A. Witkin and W. Welch. Fast animation and control of nonrigid structures. Proceedings of ACM SIGGRAPH, 24(4):243–251, 1990.