Automatic Motion Synthesis of Articulated Figures for Computer Animation

V. V. Kamat

Thesis submitted to the Goa University for
the degree of
Doctor of Philosophy.

January, 1996

Statements as required by the University

1. **Statement on originality of thesis document**

   I hereby declare that for this thesis which I am submitting for the Ph.D degree in Computer Science and Technology of the Goa university, no degree or diploma or distinction has been conferred on me before, either in this or any other university or body.

2. **Statement on the research contributions towards general advancement of knowledge**

   In this research work I have proposed a new methodology for automatic motion synthesis by the specification of motion features. Motion features have not been used before for the automatic synthesis of motion. Further I have shown with the help of an extensive implementation and a large number of experiments with virtual creatures that the methodology of motion feature specification is applicable and gives excellent results. The highly complex complex motion of gaits of legged creatures has been thoroughly investigated for successful application of this methodology. A new and efficient parallel implementation using evolutionary programming based technique has been carried out. Together all the above results have introduced a novel way of building future physically based animation systems in which the computer system will provide very high level support to the animators by doing the tedious work, leaving the animator only to concentrate on task of motion conceptualization and planning.

3. **Statement on originality of reported research**

   For this research, I have built upon various formalisms and techniques from the following disciplines, robotics, biomechanics, optimization and artificial intelligence. The proposed methodology of automatic motion synthesis of articulated figures by specification of motion features is an original contribution. Creation of the integrated simulation environment, parallelizing the stochastic population hill climbing algorithm and analyzing all the work related to gaits of legged creatures are also my original contributions of this thesis.

4. **Statement on my own individual research contributions**

   The entire work reported in this thesis has been carried out by me under the guidance of Dr. S. P. Mudur of National Centre for Software Technology, Bombay.

5. **Statement on published work in support of my candidature**

   The following publications are based completely on this research of the author.

   (a) A Survey of Techniques for Simulation of Dynamic Collision Detection and Response, Computers and Graphics , Vol. 17, No. 4, pages 379–385.

   (b) Synthesis of Realistic Motion for Legged Creatures, Software Bulletin, Vol. 3, No. 4, pages 9–16.

   (c) An Evolutionary Programming Technique Using Motion Features For Animating Articulated Figures (sent for publication).
   Co-author: Atul Jain and S. P. Mudur.

   (d) Automatic Synthesis of Gaits of Virtual Legged Creatures (sent for publication).
   Co-author: S. P. Mudur.

6. **Statement on revised version of the thesis**

   All the required modifications suggested by the thesis examiners have been incorporated in this revised version of the thesis.

[illegible]
(Dr. S. P. Mudur)
Signature of the guide

[illegible]
(V. V. Kamat)
Signature of the student

Place: NCST, Mumbai
Date: 13th December, 1996.

Automatic Motion Synthesis of Articulated Figures for Computer Animation

by

V. V. Kamat

Synopsis of the Thesis

to be submitted to

Goa University, for the award of Ph.D degree.

The study of motion

It is always fascinating to watch the motion of objects in the world, particularly the movement of living creatures. The crawl of a worm, the slither of the snake, the leap of a gazelle, the run of a dog, the gallop of a horse or the walk of a human, all look amazingly simple and beautiful, yet involve extraordinary skill in muscle coordination and balance. Reproducing even the simplest of these movements by mechanical, or other means has always been a challenge. This in part explains the tremendous appeal that Disney animations hold for young and old alike, or the effect that a moving limbed toy has on any child. Computer animation is primarily concerned with generation of the motion of virtual creatures moving around and interacting with a virtual world such that the movement appears physically realistic and is generated in a computationally efficient manner.

Apart from computer animation, motion is studied and modelled in a number of other disciplines like robotics, biomechanics, control theory and artificial life. These disciplines has their own distinct emphasis in the study of motion as discussed below, but most developments in these disciplines are bound to find application in computer animation.

Motion planning and motion control are central areas of research in robotics [22]. Forward and inverse kinematics, as well as forward and inverse dynamics are important concepts from robotics that have a direct application to the animation of articulated figures. Robots inhabit a more complex physical world than their synthetic counterpart, and must sense the environment and react to it in real time. However, the movement of robots is not just simulated but physical and is brought about by internal force generators and actual physical interaction within physical world.

Biomechanics is the study of the mechanical bases of human and animal motion [19]; with the central focus being on musculotendons which act as the main force actuators to bring about motion. Muscles have many properties that influence the type of motion produced. Biomechanics is concerned with a study of these properties as well as with the investigation of schemes for controlling muscles to yield desired motion. Traditionally, studies in biomechanics have focused on a single muscle or a single joint.

Conventional control theory addresses the problem of motion control , but of rather simple systems [57]. For more complex systems, difficulties are primarily due to the non-linearities involved. The field of control is additionally concerned with the issue of proving stability and performance under all conditions. Most systems that are studied in control theory are simple, in comparison with the motion control systems needed for typical animated figures.

Artificial life is a new science dedicated to mimicking the emergent behaviour of living systems *in silico*. Instead of trying to simply replicate the effects of living systems, artificial life researchers attempt to build these behaviours from bottom-up, much in the style that nature itself does. A typical artificial life approach begins with biological behaviour such as reproduction, evolution and locomotion and attempts to extract simple local rules behind that behaviour [104]. Many of the simulated creatures are defined by compact code, which subsequently determines the creatures' behaviour when placed in an environment. Once again most creatures being experimented with are still too simple to be interesting enough for the purposes of animation.

The movement in animation is really just an illusion; made possible because of the biological phenomenon of persistence of vision. A series of images slightly differing from each other are shown in rapid succession . The eye/mind blends them together to result in the visual illusion of movement or change. Typically

24 to 30 frames (images) per second are shown. Animation is primarily concerned with the synthesis of these images at discrete times and strictly speaking does not mandate the modelling of continuous time varying motion. However, in computer animation we find it convenient to model motion as a continuous function of time and generate the individual frames by appropriately sampling this function.

The motion synthesis problem

Movement or motion is a dynamic phenomena - it is the change in spatial configuration of an object over time. The spatial configuration of an object is defined geometrically. A simple object like a pen is completely defined by the position of a single point on its body say (x_p, y_p, z_p) and its orientation $(\theta_x, \theta_y, \theta_z,)$ in three dimensional space. More complex legged objects are defined as articulated bodies composed of links that are connected to each other via joints. However, in order to contain the complexity, in most studies rigid articulated bodies are used for modelling the motion behaviour of legged creatures.

Degrees of freedom (DOF) constitutes the minimal set of parameters needed to completely specify the configuration of a body in space. Degrees of freedom are also known as generalised coordinates. Thus the spatial configuration of the pen in 3D has 6 DOF. With just 6 parameters, the pen can be moved to any desired point and oriented in any desired direction in space. An articulated body has more DOF depending on the number of links, joint types *etc*. Consider for example the planar articulated body with 3 links and 2 rotary joints as shown in Figure 0.1. This simple body has 5 degrees of freedom. Human bodies are amongst the most complicated of articulated bodies. Typically, the human body has about 200 degrees of freedom [110].

A universally accepted convention is to consider degrees of freedom as constituting a vector and use vector notation say X to denote the spatial configuration of a body. Hence the motion of the body over a time period T would be denoted by $X(t)$, $0 \leq t \leq T$.

A rather simple definition of the synthesis problem is as follows:

> Given the geometry of an articulated body, say X, a desired type of movement say walk and a time period T, determine $X(t)$, $0 \leq t \leq T$.

The above definition hides the enormous underlying complexity in this problem. Certainly physics is involved; gravitational and other forces have to be considered. The movement of a pen dropped from a height of say, four feet from the ground is not arbitrary but completely determined by the properties associated with the pen, such as its mass, moment of inertia, coefficient of restitution *etc*. Animators take years before they acquire the necessary skills to predict the spatial configurations of objects at any time for generating a specific movement. Living creatures are active articulated bodies that can bring about their own motion through internally generated forces and interaction with environment like the wall or the ground. For such bodies, the motion synthesis problem becomes orders of magnitude more complex. It is only in very recent times with easy availability of computing power that the studies of motion of such complexity are being undertaken in the different disciplines listed above.

Traditionally, in animation the motion synthesis problem is tackled using a technique known as key-frame animation. In key-frame animation, the animator only describes a set of "Key-frames" from which the system can geometrically interpolate each DOF independently to automatically generate all the inbetween frames needed. There are two major problems with key-frame animation. Firstly it puts a large burden on the animator by requiring the adjustment of too many parameters at very fine levels of detail. For a reasonably detailed figure with 30 DOF, a minute of animation with a key-frame, say, every quarter of a second, would approximately require eight thousand values to be specified. This is perhaps an impossible task. Secondly, to generate very convincing looking motion, the animator must have a very good understanding of the motion and also possess artist like skills, for resynthesizing the internalised motion. Therefore more often than not, even after many trials, key-frame synthesized motion tends to look unrealistic and puppet like.

In the late 1980's, researchers working in the field of computer animation were convinced that if the animation has to look realistic, the physics behind the motion has to be taken into account [3, 105, 52]. This is typically done by augmenting the traditional geometric model to include other physical characteristics that computers can use to compute motion. These physical characteristics are mass of the body, its moment of inertia, external forces such as gravity, friction *etc*. The idea is to incorporate appropriate physical complexity and realism of the behaviour into the model itself, rather than requiring that it be imposed by the animator. Initial results on incorporation of physics to produce realistic movements were very encouraging.

In physically based animation, the task of synthesizing motion is accomplished in several steps. The first is to create a suitable geometric model of the articulated figure by defining the geometry of each link and its relation to the rest of the body. The second step is to supply physical parameters which include, mass, centre of mass and moment of inertia for each link. The third step is to define control parameters that will determine the necessary force/torque function which will bring about the desired motion. In the fourth step the equations of motion are assembled and solved using numerical techniques to obtain the position of the object over time. In the last step, individual images (frames) are rendered.

Currently there are two approaches in physically based animation. They typically trade off computational work for autonomy of movement versus manual work for controllability of movement.

The method of *space-time constraints* [108] and *space-time windows* [20] belongs to the first category. It poses the motion control problem in terms of trajectory through space and time which is subject to the constraints of physics and the constraints of the desired motion. This approach has close ties to keyframing. The second approach involves creating a *controller* which produces motion by directly supplying actuating forces and torques [101, 81, 49]. A parameterized controller results in a compact representation of motion. The controller is typically synthesised by searching in the multi-dimensional parameter space of urally specified in the form of desirable set of motion feature values. Further, with a given motion feature vector as input, the actual desired movement can be automatically synthesized by the use of appropriate optimization based search procedures in the space of motion controllers.

It is important to note two aspects of the problem of searching for an optimal controller for a specific movement by an articulated figure:

- The search space is large and multimodal. The number of locally optimal solutions far exceeds the useful solutions which typically occupy small portions of the search space.

- The search space may be discontinuous. Small changes in the control parameter values may lead to a large change in the fitness value.

As a consequence motion synthesis for articulated figure is compute intensive and needs efficient implementations.

Implementation and experiments

As part of this research a fairly elaborate implementation has been carried out to substantiate the above thesis. The implementation enables us to test out the automatic synthesis of motion for planar articulated bodies by specifying desirable feature values from a predefined set of motion features. The automatically synthesized motion is played back in real time on the computer display screen. Our implemenation includes the following:

1. accepts geometric definitions of an articulated body
2. models the ground and interaction with the ground
3. formulates equations of motion
4. simulates the motion for a given set of controller parameters

5. uses the desired feature values and constructs the fitness function that is used by a genetic algorithm to search for the optimal controller in the controller space.

Figure 0.2 shows an overview of the different components of our system.

Since the entire process is very demanding on computer time, the search process has been parallelized to run on a network of CPUs. The overall performance and the synthesized movements are extremely encouraging.

Thesis Organization Chapter 1 is a brief introduction to the main goal of all computer animation – the synthesis of motion of virtual objects/creatures moving and interacting in virtual environments. The importance of physical correctness and realism in synthesized motion is clearly brought out. The chapter includes a small comparative analysis of the approaches to the study of motion in computer animation and in other discipline like Robotics, Biomechanics, Artificial life and Mechanical simulations. It also gives an overview of the different approaches to the automatic synthesis of physically correct motions, the problems present in these approaches and the solution methodology proposed by us.

Chapter 2 is a comprehensive review of all known methods in computer animation for generating animated sequences involving articulated figures. Both kinematics and dynamics based techniques are discussed. The various approaches being pursued for the automatic synthesis of physically based motion are presented and motion synthesis through the automatic generation of optimal motion controllers is identified as the most promising approach to date.

In chapter 3 we discuss in detail all major aspects of optimization techniques as applicable to the motion synthesis problem. The aspects provide a framework along multiple dimensions like search space, task goal, constraints, dynamics simulation, and search algorithm, which enables us to concisely review the existing optimization methods and also any new developments that may take place in the future in this optimal motion search area.

Chapter 4 describes the importance of external object interaction in the move-

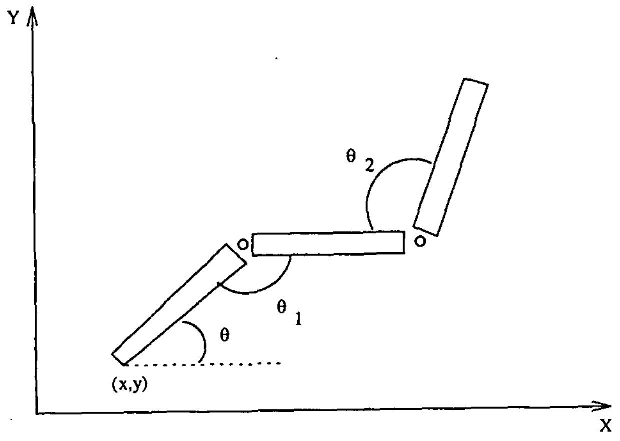


Figure 0.1: A planar articulated body with three links and two rotary joints

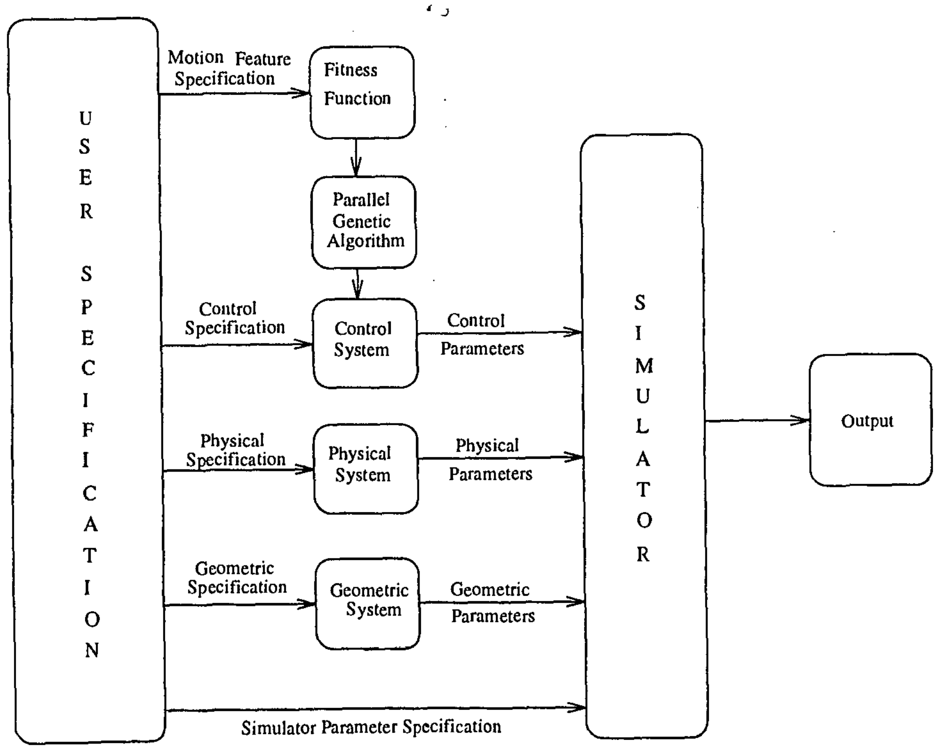


Figure 0.2: System Configuration

ment of an articulated figure. Basically all external interaction results in forces and torques that get applied to the moving figure. Collision forces are specifically the most important amongst these. The chapter reviews the collision detection and collision response problem and strategies in use for finding solutions to these problems. The different types of contact or collision like the colliding contact or the resting contact and methods for handling these are also reviewed. Finally the difficulties of modelling frictional contact are presented.

Chapter 5 addresses the primary thesis of our research – the automatic synthesis of motion through the specification of features. To begin with, we introduce the notion of motion features and formulate their specification as computable functions that take complete motions as their arguments. We formulate the performance metric that uses these feature values. The performance metric formulation is such that its value is optimal when the motion, has the specified features. Choosing the domain of gaits of legged creatures – a topic very well studied in different disciplines – we define a set of motion features that could be specified by an animator to obtain different kinds of gaits.

Chapter 6 describes our implementation and also the results from the different experiments that we conducted for synthesizing different kinds of movements for virtual legged creatures. As part of our research we have created an integrated simulation environment. The overall architecture and the different components that make up this environment such as the physical, geometric and feature model, the simulator and controller synthesizer are briefly described. Since the total computational effort involved in the motion synthesis task is excessive, we have parallelized the search process. This chapter also describes this parallel global optimal search algorithm based on evolutionary programming (a type of genetic algorithm), known as the stochastic population hill climbing (SPHC) algorithm. The parallel SPHC has been implemented using the parallel virtual machine (PVM) system. Finally the chapter describes in detail the structure of 3 virtual creatures (a single legged creature and 2 two-legged creatures), and the results of our experiments in automatically synthesising different types of gaits for these virtual creatures by the method of motion feature specification.

Figures 0.3-0.5 shows some animation sequences synthesized using our implementation.

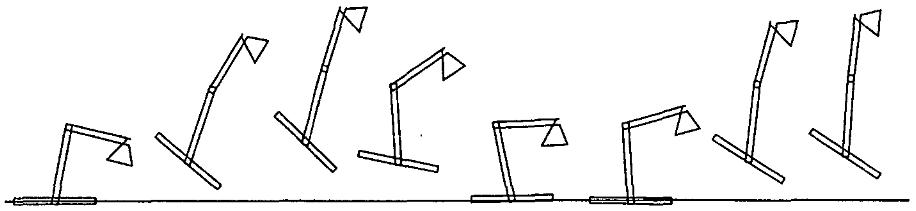


Figure 0.3: Mr. Luxo, a lamp like creature hopping

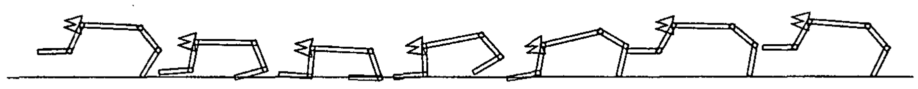


Figure 0.4: Mr. Pogo, a dog like creature walking

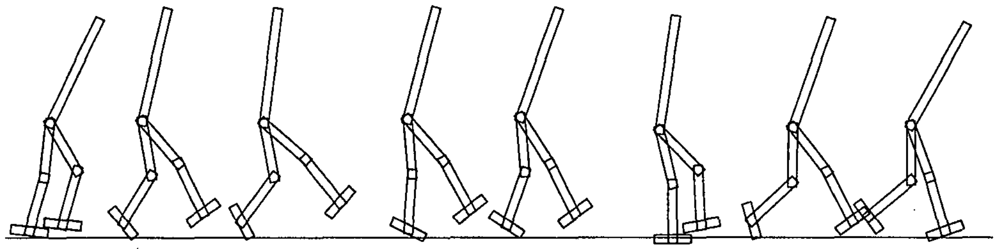


Figure 0.5: Mr. Walker, a human like creature walking

Chapter 7, the last chapter of our thesis analyses three major aspects of our work. The basic approach to solving the problem of motion synthesis, the specific solution methodology proposed in this research and the implementation and experiments carried out by us. Specifically the chapter highlights the significant contributions, some deficiencies/limitations, future extensions possible and some open problems in this area.

Acknowledgements

It would have been impossible to complete the thesis without the support of many people. First and foremost, I am grateful to Dr. S. Ramani, Director, National Centre for Software Technology (NCST) for providing an excellent research environment.

Dr. S. P. Mudur, has been more than my thesis supervisor. Not only has he been influential in shaping my thesis but he has also helped me in broadening my perspectives in looking at many things in life. His knowledge and grasp of the subject and the ability to understand things at a higher level without getting into nitty-gritty details has always intrigued me.

Special thanks are due to Atul Jain for the help in implementation, which played a crucial role in my thesis. Thanks to Bipin, Reddy and Vijay for the LaTeX support. You were always there whenever I needed help.

Thanks to all the members of the graphics division of NCST who have contributed indirectly to this thesis through their friendship, enthusiasm and discussions. I shall certainly miss your company. Pijush, your influence on me is very much there though you may not see it. Thanks to Khandge and Sakpal for video shooting the animations.

I shall fail in my duty without acknowledging the valuable support provided by the NCST library staff and systems support group. Life would not have been smooth in Mumbai without accommodation and dining facilities. Thanks to admin and canteen staff at NCST.

On a personal front I would like to thank my wife Nandini for her endless encouragement and support during my thesis work. She has given a lot of herself so that I can realize my personal goals. Nadisha, I hope you will forgive me, I know how much you missed me.

Lastly, I have a big thank you for the Goa university authorities for the leave and the financial support without which none of this would have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The movement of objects in this world has always fascinated mankind; whether it is the movement of a planet, the flight of a ball or the walk of a living creature. For centuries, philosophers, physicists and mathematicians have all attempted to study and theorize the motion of objects in the world. Thus the science of mechanics concerned with the motion and equilibrium of masses is that branch of physics which is at once the oldest and also the most fundamental, and is therefore treated as introductory to other departments of physics. It was not however until the 17th century, that a robust set of physical principles were put forward to explain the mechanics of everyday objects, when Issac Newton published his three laws of motion (in Principia) in 1687. Newtonian physics has been one of the grand success stories of science. The Newtonian paradigm has maintained its utility till date despite stunning conceptual advances from relativistic and quantum physics developed by Einstein and others. The study of Newtonian dynamics is no longer an active area of research in physics. Instead, research efforts have branched into newer disciplines such as robotics, biomechanics, physically based computer animation, and artificial life. Physically based computer animation is the primary area of focus in our research. However, we also briefly describe in this chapter the distinct emphasis that some of the other disciplines have in their study of motion. Developments in all these fields are bound to find application in computer animation as well!

1.1 Computer Animation

The primary purpose of a computer animation system is to provide assistance to the human animator in synthesizing the movement of an object such that the resulting motion appears physically correct (for example, say, obeys gravitational laws)[1], unless explicitly intended otherwise, and also conforms to the animator's goals. These goals depend on the story sequence being narrated through the animated object. If the object being animated is an autonomous character (representative of a living creature that can generate its own forces to bring about its movement) then the resulting motion should not only be physically correct, but also appear realistic [2] and natural [3]. A walk should look like a walk and be different from a hop or a run.

Typically the autonomous character is modelled as an articulated body composed of links that are connected to each other via joints. These joints have associated actuators that generate all the internal torques which along with external forces like gravity, reaction, friction etc. are used by living creatures to bring about the desired motion. While in real life the limbs are usually flexible, for simplicity, in most computer simulations the links are considered as being rigid.

Movement or motion is a dynamic phenomena – it involves change in shape and spatial configuration of an object over time. Most often, in digital simulations the shape of an object is defined geometrically. Shape being such a fundamental attribute of physical objects in the world, it has been researched extensively and the field of geometric modelling and design has evolved almost as an independent discipline [76]. If the shape of the object surface changes over time due to effects of say, forces, then these are referred to as deformable bodies, otherwise as rigid bodies.

[1]More specifically, moves in accordance with Newton's laws of motion

[2]The forces being applied are as they would be in a real world situation

[3]Appears similar to the same type of movement performed in nature by an actual living creature of the same kind.

While there is considerable ongoing research on deformable objects [96, 73], our research is primarily concerned with the movement of rigid bodies linked together, referred to as articulated bodies, and which deform only by changing the joint angles.

The movement in animation is really just an illusion, made possible because of the biological phenomenon of persistence of vision. A series of images slightly differing from each other are shown in rapid succession. The eye/mind blends them together to result in the visual illusion of movement or change. Typically 24 or 30 frames (images) per second are shown. Animation is primarily concerned with motion synthesis, which is the creation/presentation of these images at discrete times. Strictly speaking therefore it does not mandate the modelling of continuous time varying motion. However, in computer animation we find it convenient to model motion as a continuous function of time. This function is then appropriately sampled in order to generate the individual frames showing spatial configurations of the creature(s).

The spatial configuration of an object is defined geometrically. A simple object like a pen is completely defined by the position of a single point on its body say (x_p, y_p, z_p) and its orientation $(\theta_x, \theta_y, \theta_z,)$ in three dimensional space. *Degrees of Freedom* (DoFs) constitute the minimal set of parameters needed to completely specify the configuration of a body in space. Degrees of Freedom are also known as generalized coordinates. Thus the spatial configuration of the pen in 3D has 6 DoFs. With just 6 parameters, the pen can be moved to any desired point and oriented in any desired direction in space. Depending on the number of links, joint types etc., an articulated body would have many more DoFs. Consider for example the planar articulated body with 3 links and 2 rotary joints as shown in Figure 1.1. This simple body has 5 degrees of freedom. Human bodies are amongst the most complicated of articulated bodies, with about 200 degrees of freedom [110]. A universally accepted convention is to consider degrees of freedom as constituting a vector and to use vector notation say X, to denote the spatial configuration of a body. Hence the motion of the body over a time period T would be denoted by $X(t)$, $0 \leq t \leq T$. This is also referred to as the trajectory.

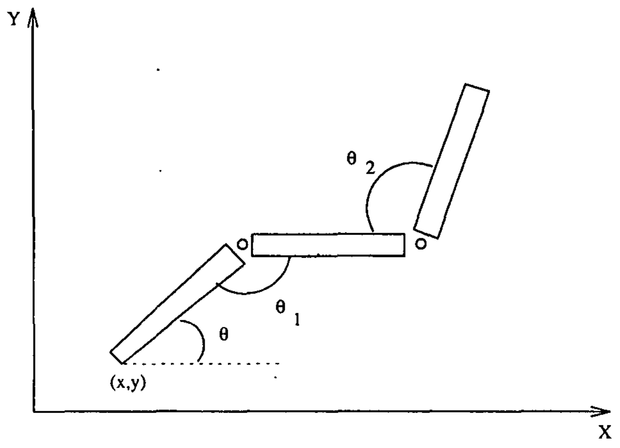


Figure 1.1: A planar articulated body with three links and two rotary joints

A simple definition of the motion synthesis problem is thus as follows:

Given

- the geometry of an articulated body say X,
- a desired type of movement say, walk, and
- a time period T,

determine X(t), $0 \leq t \leq T$.

The above definition hides the enormous underlying complexity in this problem. Certainly physics is involved; gravitational and other forces have to be considered. The movement of a pen pushed from a table of height of say, four feet from the ground is not arbitrary but completely determined by the properties associated with the pen, such as its mass, moment of inertia, coefficient of restitution etc. and the initial force applied. Animators take years before they acquire the necessary skills to predict the spatial configuration of objects at any time instance for generating a specific movement. For living creatures, which bring about their own motion through internally generated forces and interaction with environmental objects like the wall or the ground, the motion synthesis problem becomes orders of magnitude more complex. It is only in very recent times with easy availability of computing power that motion synthesis tasks of such complexity are being undertaken [54].

There is another view to the motion synthesis problem. All possible motion paths for a given character can be considered as forming a space of trajectories. And the problem of synthesizing a particular motion can be viewed as that of searching for a suitable trajectory in that space. The trajectory that is finally selected must satisfy all the physical requirements and also the animator's goals. Consider the example of synthesizing the trajectory of the pen pushed from the table. (cf Figure 1.2.) From the space of trajectories that *specific*

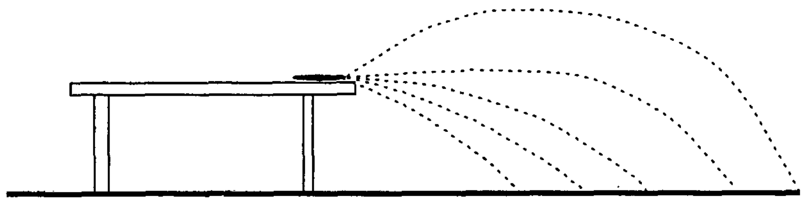


Figure 1.2: Some possible trajectories for a pen pushed off a table

trajectory has to be chosen which accelerates as determined by the initial force and also by the gravitational force acting on it. Search for this kind of trajectory is comparatively simple and can be totally automated using what are being called as *physically based animation* techniques [10]. For autonomous articulated figures however, the search problem gets extremely complex. Firstly the number of DoFs is very large. As a result the trajectory space is of very large dimension. Secondly, there is always built in task level redundancy, i.e any behavioural goal can be achieved in many different ways. Thus, for example, a cup of coffee might be reached while moving the hand along many different paths. Usually the search is cast as a non-linear constrained optimization problem. The physical laws, and physical and user specified constraints are to be satisfied while the animator's goals are in the form of an objective function that has to be optimized.

Whatever may be the view, whether it is one of defining or creating a suitable trajectory or that of searching for the most suitable trajectory, the problem of synthesizing the movement of a virtual object or creature inhabiting and interacting with a virtual world consisting of other virtual objects or creatures remains the fundamental problem of computer animation. All research in computer animation is thus oriented towards new techniques that will enable the development of software tools that will assist the animator in synthesizing different movements for different types of objects/creatures. Over the years a variety of techniques have evolved, ranging from providing the animator with simple tools for interactive drafting, painting and trajectory interpolation to the very highly sophisticated tools, that enable embedding of complex motion behaviour into the virtual object or creature. The focus has completely shifted from simple transformation based movements of single or groups of objects [78] to the complex movement of legged creatures [33, 87].

While in chapter 2 we shall present a more detailed and comprehensive review of the current state of the art in motion synthesis for articulated figures, below, we briefly describe a number of other disciplines in which motion synthesis is of concern.

Robotics

The construction of autonomous legged robots is one of the goals of robotics. This involves creating systems that can sense their environment and can travel in a obstacle filled environment. The two central issues that are of importance in robotics are that of motion planning and motion control. Motion planning [66] is typically treated as a geometric problem of obstacle avoidance where as motion control [14, 86, 68] involves design of a control system.

Biomechanics

Biomechanics is the study of the mechanical bases of biological activity [106, 107]. One of the primary goals of biomechanics is to understand the mechanisms in human locomotion so that it is possible to have a better design of prosthesis for disabled persons. In this and other areas dealing with human limbs and their substitutes, a consensus is that the end (artificial) products should duplicate the performance of their biological counter parts as closely as possible.

Artificial life

In Artificial life, the goal is to mimic the behaviour of living systems *in silico* [62, 104]. Instead of trying to simply duplicate the effects of living systems, artificial life researchers attempt to build these behaviours from bottom-up, much in the style that nature itself does. A typical artificial life approach begins with biological behaviour such as reproduction, evolution and locomotion and attempts to extract simple logical rules behind that behaviour. Many of the simulated creatures are defined by compact code, which subsequently determines the creatures behaviour when placed in an environment. Most creatures being experimented with are still too simple.

Mechanical Simulations

For years, dynamic analysis [82, 42] programs such as DADS and ADAMS have incorporated graphical post-processors capable of displaying the motion of simplified models. The major benefit of such programs is that they allow the designer to quickly build an electronic prototype and test it to see how the moving parts will function in the real physical world. The recent work by Hodgins *et. al* [49] is one similar such application. They have applied dynamic simulation to a platform diver to study how changes in technique can affect the diver's performance. Through motion simulation they could visualize what a dive might have looked like if the athlete had opened up earlier or later, or had left the diving platform differently.

1.2 A Relative Comparison of Motion Synthesis Studies

As can be seen from the brief descriptions of the different disciplines above, it is clear that motion synthesis is also a problem of interest in disciplines other than computer animation. While there are similarities in the studies, there are some fundamental and major differences in problem scope, research traditions and practical requirements. We elaborate on this in detail below.

Motion synthesis in artificial life is the closest to that in computer animation. In both areas virtual creatures have to interact and move in virtual worlds. However the emphasis in artificial life is on embedding mechanisms that enable learning of locomotion behaviour, and this along with all other kinds of performance behaviours. As of today, in this field movement goals are simple and are essentially self determined by the virtual creatures rather than imposed externally as done by animator. Real-time movement and response to collisions in the environment are additional constraints that make it very difficult at this stage to consider the embedding of highly complex external goal oriented motion behaviour. In the present state of art, the kinds of movements being synthesized are extremely simple as compared to what is desired for animated creatures. Moreover there is no requirement that the movements of these artificial creatures must appear realistic or natural.

Mechanical simulations, Biomechanics and Robotics are all concerned with the motion of real physical objects. Mechanical simulations deal with the motion of rigid linkages primarily towards understanding/analyzing their motion behaviour under different load conditions. There is certainly no explicit concern in any way to have these linkages represent, the limb structures of living creatures, though there have been some specific efforts towards simulating very specific movements like walking, hopping and diving. Biomechanics, while it is concerned with the motion behaviour of living creatures, is really concerned primarily at the individual limb and muscle level.

Motion planning and control in robotics once again comes rather close to motion synthesis in computer animation. There are however significant differences. In particular

1. The robot's linkage structure is physically real and it inhabits, interacts and moves in a real world. Unlike in computer animation, no idealized simplification is possible either for the parts of the robot or for the objects in the environment. Actuator forces, external interactions like collision and external forces like gravity, friction, reaction etc. are all real. As such, robots inhabit a very much more complex world than their virtual

counterparts in computer animation.

2. On the other hand physical correctness is built in. A robot will not move through a wall, nor can its motion trajectories be not in accordance with the laws of physics. As such classical motion planning in robotics eschews physics totally. Since physical realisability of the synthesized motion is usually not of concern, the emphasis in robot motion synthesis is on obstacle avoidance, treated more as a geometric problem. Such an approach is clearly not appropriate for motion synthesis in computer animation.

3. An animated sequence may involve several characters with novel physical characteristics. Thus an animator would need to rapidly synthesize a variety of complex motions for one character and for many characters with different physical attributes. Where as current studies in robotics are primarily concerned with a single robot or a class of robots with similar physical attributes.

4. An autonomous robot must be able to synthesize its motion trajectory independently. This is especially challenging in obstacle filled environments. In contrast an animator can afford to build up a character's motion trajectory piecemeal, concatenating a sequence of trajectories to obtain a final complex motion that avoids obstacles and behaves in conformance with the interaction with objects in the virtual world. Basically the animator and the computer can carry out the motion synthesis task in a collaborative manner.

5. Finally, in robotics real-time response is essential, and this puts very heavy demands on the computational resources that have to be built in. On the other hand in computer animation the synthesized motion is played back and has only to appear physically correct, realistic and natural. Computations taking time of the order of a few hours or days for synthesizing the motion of a few seconds are quite acceptable. There are no inherent demands on the kind of computing power that needs to be available. Also while physical correctness is a goal, appearance is more important and slight deviations not noticeable to humans can easily be permitted.

1.3 The Physical Basis in Motion Synthesis

Initial computer animation research can be dated back to early 1960's. The approach was purely geometric in nature. The responsibility lay entirely with the animator for the resulting motion to look physically correct, realistic and natural. The demands of physically correct behaviour had to be understood and imposed explicitly by the animator. Interactive tools were made available to assist in this task so that the animator could rapidly alter the synthesized motion at a local or global level and rapidly make the necessary number of trials before choosing the final trajectory for the object or character. By the late 1980's however, researchers working in the field of computer animation were convinced that if the animation has to look realistic, the physics behind the motion has to be taken into account and thus *physically based animation* was introduced. This is typically done by augmenting the traditional geometric model to include other physical characteristics that computers can use to compute motion. These physical characteristics are mass of the body, its moment of inertia, external forces such as gravity, friction etc. Interaction with other objects in the environment and resulting behaviour is also modelled and a variety of collision detection and collision response techniques have evolved [53]. The idea is to incorporate appropriate physical complexity and realistic behaviour into the model itself, rather than imposing it on the animator. Initial results on incorporation of physics to produce realistic movements were very encouraging [3, 105, 52].

However, this is not without problems. As we shall see later in chapter 2, the specification of forces and torques to produce any desired motion is non-intuitive and certainly non-trivial. Further, once time varying forces and torques are specified, the motion is completely determined and is autonomous, and not any more under the control of the animator. Thus incorporation of physics into the model results into loss of fine control that an animator always needs to have over the generated motion. A number of techniques have therefore evolved to accept control specifications for desired motion in a more indirect manner but with adequate automatic methods built in for deriving the forces and torques that need to be applied [108, 100, 81, 38].

Thus in physically based animation today, the task of synthesizing motion is accomplished in several steps. These are listed below.

1. Create a suitable geometric model of the articulated figure by defining the geometry of each link and its relation to the rest of the body.
2. Supply physical parameters which include, mass, centre of mass and moment of inertia for each link.
3. Define control parameters that will enable automatic determination of necessary forces and torques which will bring about the desired motion.
4. Assemble the equations of motion and solve them using numerical techniques to obtain the position of the object over time.
5. Render the individual images (frames).

Of the five steps, except for the third, all others are very well studied and excellent working solutions exists [22, 29]. Step 3 however poses rather difficult problems.

Currently available methods trade off manual work for controllability versus autonomy for physical correctness. For example, there are methods to control at a low-level by interpolation of poses with adherence to physical constraints [15, 108]. There are also methods that expect only high level goal specification such as "Jump as high as possible" or "Walk as fast as possible" [81, 100]. Most of these problems are highly under constrained and use some kind of optimization in order to find a solution. Here, a critical problem lies in the specification of the performance metric. Performance metrics are very indirect ways of specifying motions and lack any immediate intuitive associations with the desired motions.

Typical performance metrics used are like minimization of external energy, or travel maximum distance etc. To associate such metrices with desired motions such as walk, jump, hop etc. is not very straight forward and involves lot of trial and error. As a result, physically based animation is still in the research laboratories. Animators find it more convenient to use purely geometry based techniques taking full responsibility for physical correctness and realism, while keeping precise control over the synthesized motion. This is of course at the expense of the considerable manual efforts that the animator has to put in. What is really needed are high level physically based motion synthesis techniques that require specifications which are easy, highly intuitive and at the same time enable the animator to have any desired level of control over the generated motion. In short, more the automation the better it is, provided complete control is in the hands of the animator. This research has primarily addressed this problem and has proposed an innovative and implementable methodology based on the use of motion features.

1.4 Thesis Statement

The different types of movements that we see in the real world have their own distinct attributes or features that uniquely characterize them. A run is different from a walk; which is different from a jump. For example in a walk at least one foot is always on the ground at any time; where as in a jump both feet can be away from the ground. Similarly its duty cycle, maximum height from the ground and so many other attributes are different from that of a jump. We refer to these distinguishing attributes as *motion features*. Clearly, we humans are capable of recognizing motion features that enable us to distinguish amongst different types of movements.

Mathematically speaking we represent a motion by the use of a feature vector $\mathbf{f} = (f_1, f_2, \ldots, f_n)$, where $f_1, f_2, \ldots, f_n$ are the n individual features. Each feature is a computable function which when applied to the given motion returns a numerical value(s). Thus for a given motion $X(t)$, $f_i(X(t))$, $0 \leq t \leq T$ denotes the ith feature value. The set of motion features forms a feature space. In feature space, motions of the same kind and for the same body cluster together. Different kinds of motion result in different clusters and these clusters are separable.

Our main thesis can now be stated as follows:

> Motion features are quantifiable attributes of different types of movements and enable distinguishing amongst them. The task of motion synthesis of active articulated bodies for computer animation is specified in the form of a desirable set of motion feature values. Further, with a given motion feature vector as input, the actual desired movement can be automatically synthesized by the use of appropriate optimization based global search procedures.

As part of this research a fairly elaborate implementation has been carried out to substantiate the above thesis. Using this implementation we have carried out the automatic synthesis of motion for planar articulated bodies by specifying desirable feature values for a predefined set of motion features. The automatically synthesized motion is played back in real time on the computer display screen. Our implementation simulates the motion for a given set of control parameters and uses the desired feature values to analytically formulate the fitness function that is used by an evolutionary programming algorithm to search for the desired motion.

Since the entire process is very demanding on computer time, the search process has been parallelized to run on a network of CPUs resulting in considerably reduced elapsed times for searching. The synthesized movements appear highly realistic and natural and the overall performance is extremely encouraging.

1.5 Thesis Organization

Chapter 2 is a comprehensive review of known methods in computer animation for generating animated sequences involving articulated figures. Both kinematics and dynamics based techniques are discussed. The various approaches being pursued for the automatic synthesis of physically based motion are presented and motion synthesis through the automatic generation of optimal motion controllers is identified as the most promising approach to date.

In chapter 3 we discuss in detail all major aspects of optimization techniques as applicable to the motion synthesis problem. The aspects provide a framework along multiple dimensions like search space, task goal, constraints, dynamics simulation, and search algorithm, which enables us to concisely review the existing optimization methods and also any new developments that may take place in the future in this optimal motion search area.

Chapter 4 describes the importance of external object interaction in the movement of an articulated figure. Basically all external interaction results in forces and torques that get applied to the moving figure. Collision forces are specifically the most important amongst these. The chapter reviews the collision detection and collision response problem and strategies in use for finding solutions to these problems. The different types of contact or collision like the colliding contact or the resting contact and methods for handling these are also reviewed. Finally the difficulties of modelling frictional contact are presented.

Chapter 5 addresses the primary thesis of our research – the automatic synthesis of motion through the specification of features. To begin with, we introduce the notion of motion features and formulate their specification as computable functions that take complete motions as their arguments. We formulate the performance metric that uses these feature values. The performance metric formulation is such that its value is optimal when the motion, has the specified features. Choosing the domain of gaits of legged creatures – a topic very well studied in different disciplines – we define a set of motion features that could be specified by an animator to obtain different kinds of gaits.

Chapter 6 describes our implementation and also the results from the different experiments that we conducted for synthesizing different kinds of movements for virtual legged creatures. As part of our research we have created an integrated simulation environment. The overall architecture and the different components that make up this environment such as the physical, geometric and feature model, the simulator and controller synthesizer are briefly described. Since the total computational effort involved in the motion synthesis task is excessive, we have parallelized the search process. This chapter also describes this parallel global optimal search algorithm based on evolutionary programming (a type of genetic algorithm), known as the stochastic population hill climbing (SPHC) algorithm. The parallel SPHC has been implemented using the parallel virtual machine (PVM) system. Finally the chapter describes in detail the structure of 3 virtual creatures (a single legged creature and 2 two-legged creatures), and the results of our experiments in automatically synthesising different types of gaits for these virtual creatures by the method of motion feature specification.

Chapter 7, the last chapter of our thesis analyses three major aspects of our work. The basic approach to solving the problem of motion synthesis, the specific solution methodology proposed in this research and the implementation and experiments carried out by us. Specifically the chapter highlights the significant contributions, some deficiencies or limitations, future extensions possible and some open problems in this area.

Chapter 2

Animation Techniques for Articulated Figures: A Review

There are two basic approaches used in synthesizing the movements of articulated figures. These are the kinematics and the dynamics based approaches.

In the approach based on kinematics, we deal with motion without consideration of mass and forces which cause motion. Within the science of kinematics we largely study the position, velocity, acceleration and all higher order derivatives of the position variables with respect to time. Thus the kinematics approach addresses all the geometrical and time based properties of the motion. Traditional computer graphics animation techniques are all based on the kinematics approach. The more important kinematic methods that have evolved for motion synthesis in computer animation are key-frame animation, direct user manipulation, geometric constraints and morphing.

In the approach based on dynamics, we deal with motion based on mass, inertia etc. and under the influence of forces and torques, in accordance with Newton's laws of motion. For a collection of bodies, each body's motion is only due to the forces and torques acting directly on it; interaction between bodies is mediated by forces and torques. A fundamental requirement of the dynamics approach is the synthesis of force and torque functions required to cause any animator desired motion. For example, in order to accelerate an articulated figure from rest, glide at constant velocity, and finally decelerate to a stop, a complex set of force and torque functions must be applied at the joints. The exact form of the required force or torque functions depends on the spatial and temporal attributes of the path taken by the figure; as well as the mass properties of the links, friction in the joints etc. Dynamic equations of motion are used to simulate the movement. This is done by reformulating the dynamic equations so that acceleration is computed as a function of force or torque. Dynamic equations of motion are also used in the derivation of these torque functions needed to cause the figure to follow a desired path. The dynamics approach in computer animation though relatively recent is being aggressively researched world wide by a large number of groups. This includes techniques such as direct force control, motion controllers, dynamic constraints, spacetime constraints and automatic motion controller synthesis.

In the rest of this chapter we briefly discuss representation of articulated figures, trajectory computation, and forward kinematics computations. We then review all the important kinematics and dynamics based techniques listed above.

2.1 Articulated Figure Representation

The articulated figure is represented using a set of rigid links, arranged in a tree structure (cf Figure 2.1). Each link of the body possesses one joint at which it is attached to its parent link and may possess one or more joints at which child links are attached. The links move relative to each other depending upon type of the joint. The type of joint present between adjacent links will determine the allowable number of DoFs between them. For example a pin joint will allow 1 DoF, cylindrical joint will allow 2 DoF, ball and socket joint 3 DoF, etc. (cf Figure 2.2).

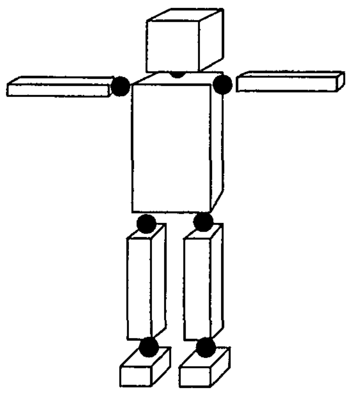


Figure 2.1: Tree structured human like figure

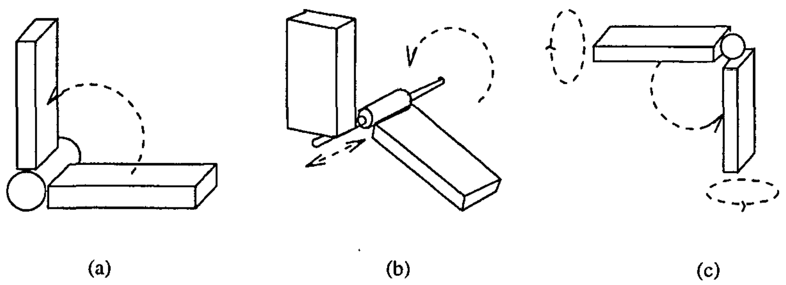


Figure 2.2: (a) Pin joint with 1 DoF (b) Cylindrical joint with 2 DoF (c) Ball and socket joint with 3 DoF.

2.2 Trajectory Generation

In animation we are always concerned with the location in time of each link of the articulated figure. This is the purpose of trajectory generation. A common way of causing a figure to move from one place to another in a smooth controlled fashion is to cause each joint to move as specified by a smooth function of time. Usually each joint starts and ends its motion at the same time so that the figure motion appears coordinated. Exactly how to compute these joint motion functions is the problem of trajectory generation. All animation techniques must finally produce these joint angle functions with respect to time defined for each of the joints of the articulated figure. These functions are then sampled, typically at the rate of $1/24^{th}$ of a second or so, to obtain the individual frames of the figure that need to be rendered for playback.

2.3 Forward Kinematic Computations

The next basic problem is that of forward kinematics. This is the static geometrical problem of computing the position and orientation of each of the links. This problem can be represented as

$$\mathrm{X} = T(\Theta)$$

where X represent the Cartesian coordinates and Θ represents the joint angles. This can also be thought of as transforming the representation from a joint space to a Cartesian space. This is typically done using Denavit-Hartenberg (DH) [25] notation.

According to DH notation, kinematics of each link is described relative to its neighbor by attaching a coordinate frame to each link (cf Figure 2.3). The representation uses a set of four parameters that are used to define a linear transformation matrix between adjacent coordinate systems. The four parameters are the length of the link a, the twist of the link α, the distance between links d, and the angle between links θ. The transformation between link $i-1$

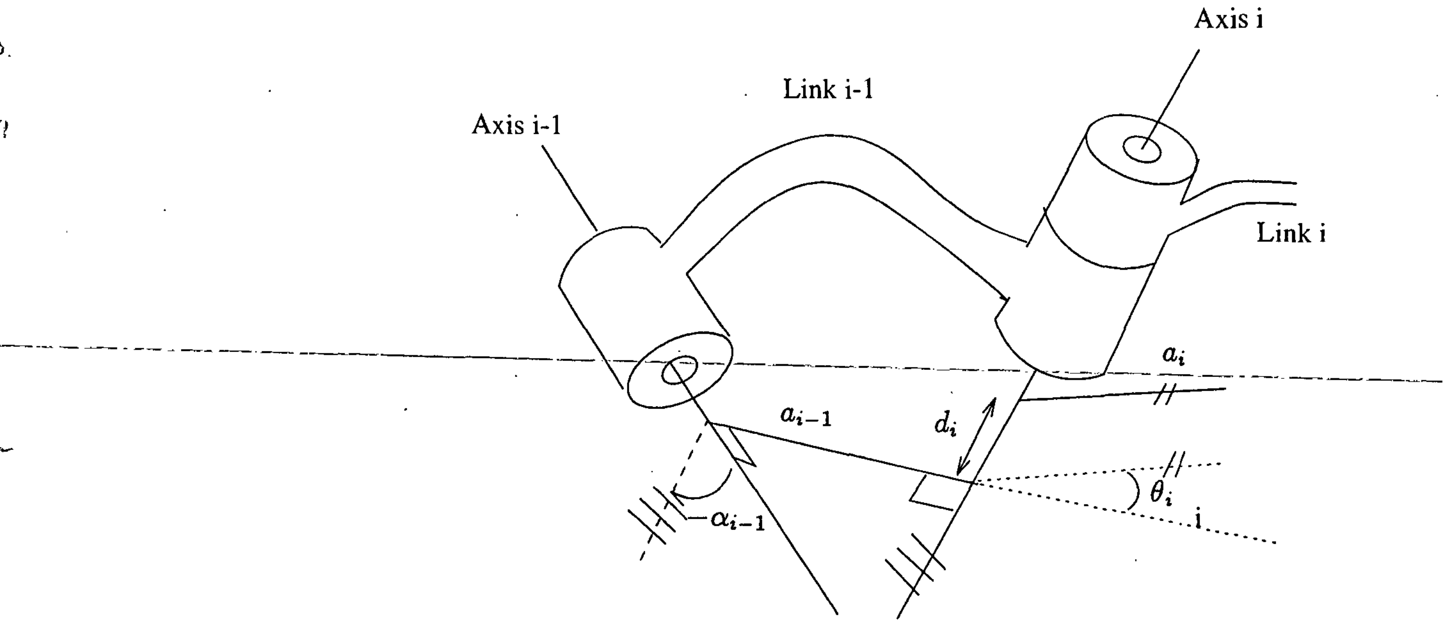


Figure 2.3: Geometric relation between two links, DH notation

and link i (cf Figure 2.3) is defined by

$$^{i}T_{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i\cos\alpha_{i-1} & \cos\theta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} & -sin\alpha_{i-1}d_i \\ \sin\theta_i\sin\alpha_{i-1} & \cos\theta_i\sin\alpha_{i-1} & \cos\alpha_{i-1} & \cos\alpha_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Though the general transformation matrix looks quite complicated, in practice, if we consider only one or two types of joints, the matrix gets considerably simplified. For example, if we consider only planar articulated figures with rotary joints, then $\alpha_i = 0$ and $d_i = 0$.

If an articulated figure's position at any instant of time is given by a set of joint angles $\Theta = (\theta_1, \theta_2, \ldots \theta_n)$, the computation of the figure's position in the world coordinate system is done in a straight forward manner by applying successive matrix multiplications between adjacent links, starting at the base of the link. Each link is transformed from its own coordinates to the world coordinate space.

2.4 Kinematics Based Animation

2.4.1 Key-frame Animation

One of the earliest and simplest kinematic method is the key-frame animation technique derived directly from the manual cel based method. The cel based method is basically for 2D animation. Flat images are hand-drawn and painted one character at a time on transparent sheets known as cels. Foreground and background parts of the image are on separate cels. Each image of an animation sequence is then composed by stacking the cels in the required order so that foreground objects and characters are overlaid over background cels. While this is very time consuming, cel animation has produced spectacular results. Any animated Disney film is a fine example of this. Increasingly computer graphics techniques are used for improving the sketching, inking and colouring process in cel animation.

In the key-frame system, the animator need not describe each frame. Instead the animator describes a set of "key-frames" from which the animation system

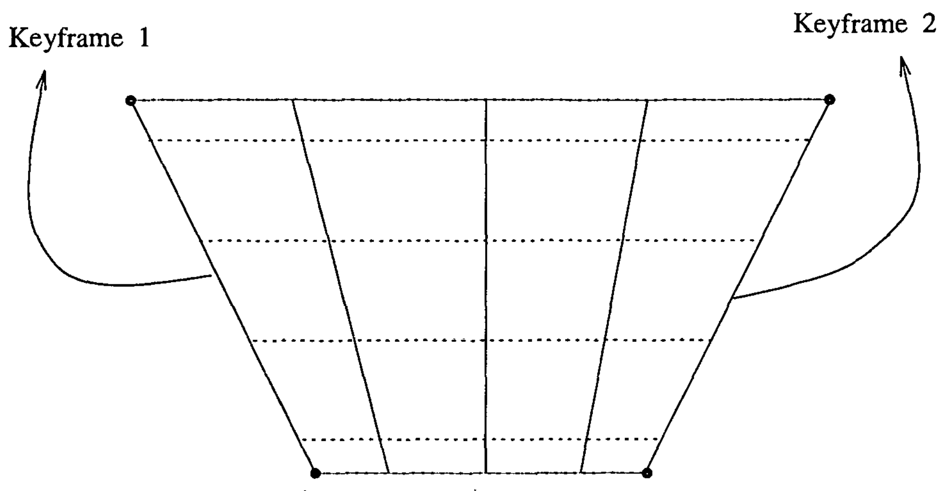


Figure 2.4: Length distortion due to linear interpolation between key-frames

can geometrically interpolate each DoF, to obtain the parameter values for each in between frame. Linear interpolation is very easy to implement but may lead to distortion (cf Figure 2.4) and jerky motion due to velocity discontinuities at the key-frames. There are more sophisticated methods based on splines which make use of cubic or higher order interpolation to generate smoother motion [60, 90, 94].

2.4.2 Direct User Manipulation

The major problem with key-frame animation is that, for articulated objects having large number of DoFs, specifying the key-frames is very tedious. For example, consider the motion specification problem of an articulated object such as a human or an animal. For a reasonably detailed figure with 30 DoFs, a minute of animation with a key-frame every quarter of a second, would require approximately eight thousand values to be specified.

In the key-frame animation technique each DoF is independently interpolated. Coordinated movement of links is entirely the responsibility of the animator. The motor control program technique suggested by Zeltzer [109] to overcome this problem is one of the early kinematic methods in which the joint angle variation function is directly specified for each joint of the articulated figure. No interpolation is involved. Instead a procedure referred to as a motor program is associated with each joint of the figure. Motor programs use hierarchical control structures for coordinated movement among several joints in a limb. One motor program can thus control several DoFs. However, its main disadvantage is that the entire motion has to be preprogrammed. There is no assistance provided by the system to formulate these motor programs. Embedding complex motion behaviours in the form of motor programs is extremely difficult.

2.4.3 Inverse Kinematics

Another very popular method to reduce the number of parameters that an animator has to specify is using inverse kinematics. Inverse kinematics takes advantage of the fact that we are often interested in only the end position and orientation of an articulated chain. For example the end of a limb may be required to be moved to a specific position. Since we know the goal to be achieved, we can, in effect apply functional constraints so that arm and hand linkages can be controlled with fewer parameters. The inverse kinematic methods try to determine the sequence of intermediate joint angles that will place the limb in the correct position. The inverse kinematics problem can be stated as that of finding the joint angles Θ of the links given the position and orientation of the last link X. Given that animals have many degrees of freedom, the problem of finding the joint angles that correspond to a given limb location and orientation does not have a unique solution. Figure 2.5 shows one example of this.

Typically inverse problems are solved by optimization [55]. The system is constrained sufficiently such that the number of possible configurations are reduced. For example, one such form of constraint is a limit on the range of angles that a joint can move. For multiple constraints, where it may not be possible to satisfy exactly all the constraints, Badler *et al.* [6] attempt to minimize the deviation from these multiple constraints or goals by assigning weights that act as priorities for goals. They use a tree structure in which the nodes are goals and balancing the tree is equivalent to minimizing the

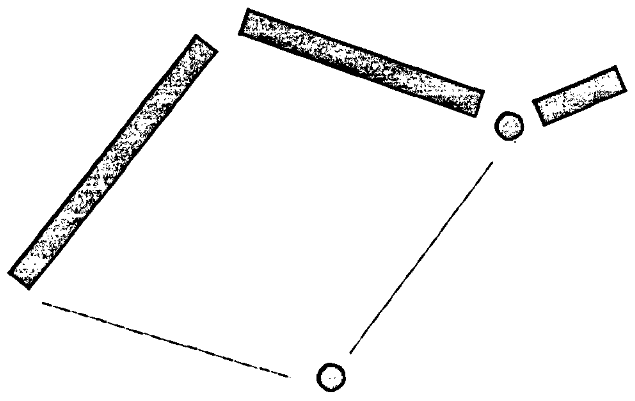


Figure 2.5: Three-link figure (dashed lines indicate a second solution)

total deviation from all the goals. Similarly, Girad [34] has used dynamic programming to reparametrize the trajectory with respect to time under the optimization criteria of jerkiness minimization to create a smooth motion. As we said earlier, the problem with inverse kinematics is that many a times the solution is not well defined and optimization methods have to be used.

2.4.4 Morphing

The last few years have seen a very rapid increase of the use of synthetic 3D characters in animation films. These synthetic characters, ranging from snakes, dinosaurs to human figures, interact and move in a 3D world. Each image frame is photorealistically rendered. One of the main requirements in such 3D animation is, therefore deformation of the shape of a 3D character as it moves. As such all the above kinematic techniques are extremely difficult to use for this kind of 3D animation. The most popular kinematic technique in use for such animation is *morphing*. Morphing in some sense is very similar to key-frame animation. In morphing, the animator has to compute the in-between frames to do a metamorphosis between two different objects. It basically involves altering the surface description of one object to map into other. The objects involved may be very similar in type, for example people's faces, (all faces have the same components) or they may be as different as cube and a sphere. In cases of similar topology, the morphing will be point to point mapping of some type of mesh created on the surface of each object. When using morphing for synthesizing movement of a synthetic character, mesh level interpolation is done between two poses of the same character. For realistic and smooth movement effects, these poses have to be chosen very carefully. Usually a very large number of poses are needed for articulated figures with large number of DoFs. This is extremely tedious and time consuming.

2.5 Drawbacks of Kinematics Methods

The major strength and also the problem of the kinematic approaches, particularly key-frame and morphing is that the methods give complete control to the animator over the synthesized motion. The animator as a result is usually very comfortable with these techniques and that largely explains their popularity to date, in spite of the fact that the amount of effort and data needed for using these methods is enormous. The problem with an animator having this kind of complete control over the trajectory is that an animator can produce motion that is physically unrealizable. Hence the attributes of physical correctness, realism and natural appearance have all singularly to be the concern of the animator. Thus these techniques require extremely skilled animators, who over years of observation and practice have understood the time and physical complexity of motion and are able to translate that into the computational model of interpolated key-frame images. As a result complex 3D animation is inordinately expensive and attempted only by few. And very often, key-framed or morphed animation tends to look unrealistic and puppet like.

In spite of these drawbacks some of the best animations today are results of these kinematic techniques. Motion capture devices are flooding the market. Real life motion is enacted and a large number of different types of sensors are attached to the body of the real character and the motion trajectories of all the important parts are digitally recorded. This recorded motion is then applied brute force to the synthetic character after some simple local manipulations that may be necessary. While these methods have been used for some very spectacular 2D animation, simpler, less labour intensive and more efficient methods are essential if 3D animation has to be widely used [1].

2.6 Dynamics Based Animation

In animation methods based on dynamics, the traditional geometric models are augmented to include other physical characteristics that computers can use to compute motion. These physical characteristics are mass of the body, its moment of inertia, external forces such as gravity, friction etc. Of course the added realism is not free, it is at the cost of increased computational complexity.

In physically based animation, the object's behaviour over time is modelled as a continuous function determined by the equations of motion. The individual frames are generated by sampling this function at an appropriate rate. This is known as forward dynamics simulation. Forces and torques are the main agents which bring about the motion. There are a number of methods to formulate and solve the equations of motion. We shall discuss some of these methods in detail in chapter 3. Synthesizing motion using any dynamic based method implies an underlying method that generates the forces and torques as functions of time that need to be applied for obtaining the desired movements of objects. Once these force and torque functions are derived then all further stages of formulating the differential equations of motion and then solving them for simulation is rather straight forward. However, the problem of composing the right force or

[1]While computer imaging is considered an indispensable production tool for all Hollywood films today, it should be recognized that much of the use of this technology is not for animation but for imaging effects with titles and other static imagery. The advertising and publicity that is usually associated with films that have used some computer animation might tempt one to believe that 3D character animation is used extensively and hence is now only an issue of using the right technology. On the contrary, even in the most spectacular examples of digitally created cinematic imagery to date i.e Terminator 2, Jurassic park, Apollo 13 and Casper to name a few, the computer generated sequences make but a fraction of the running time (ranging from 6 minutes in Jurassic park to about 40 minutes in Casper). The only film that has 100% 3D character animation is the new film, Toy story, which has a running time of about 77 minutes. The film made extensive use of motion capture and 3D morphing. The software required is said to have consumed 300 Mbytes per frame, produced by 117 Sun SPARC 20s, took four years to make and required 800,000 machine hours just to produce the final cut [91].

torque functions in the presence of external forces like gravity, friction, reaction from collisions etc. so as to result in a movement that achieves a coordinated goal or expresses a certain desired quality of movement is an extremely difficult one. All the different dynamics based methods being developed are essentially trying to solve this problem so that an animator can easily provide the right specifications to obtain the right torque or force functions.

2.6.1 Explicit Force and Torque Specification

Some of the early dynamics based methods were rather simple. The animator was expected to directly specify the forces or torques that need to be applied at individual joints as functions of time. These functions were then used to simulate the motion of the object(s). The resulting motion was viewed by the animator and then the force or torque functions had to be manually tuned in an iterative manner until the force torque functions resulting in the desired motion were obtained. This *explicit control* gives physically correct trajectories, but the level of automation is low. Too much of effort is required to discover and refine acceptable motion. This is primarily because force or torque functions, unlike trajectory functions, are completely non-intuitive and hence unnatural to specify directly. The indirection is introduced by the differential equations of motion and even physicists deeply involved in the study of dynamics would find specification of these force or torque functions for a desired motion not at all a simple task. Animators who are more artists than physicist or engineers thus normally find this form of specification not only unnatural but also very inconvenient and difficult. This becomes quite obvious when we consider the nature of the force function that needs to be applied for making, say, a dog like creature walk on the ground. Figure 2.6 shows the torque function that has to be applied to joint labeled 4.

Yet another aspect of the movement of active bodies that makes this method of explicit specification of force or torque functions extremely complex is as follows:
living creatures make use of external forces resulting from interaction in order

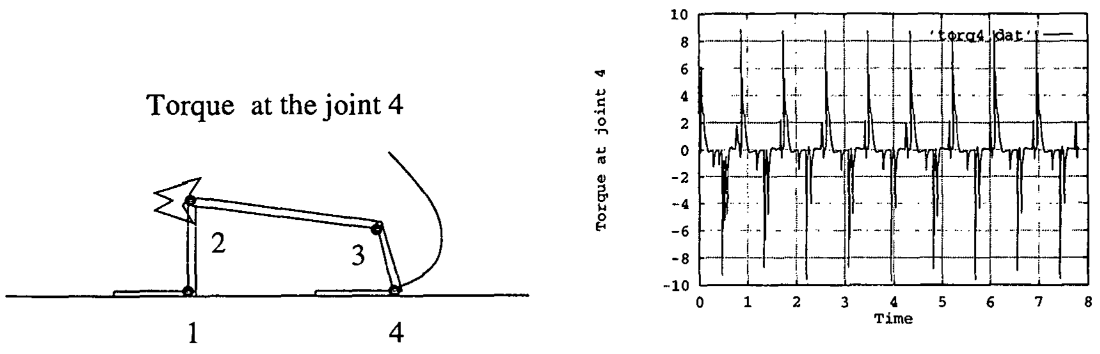


Figure 2.6: Sample torque at a joint for a walking dog like creature

to locomote themselves. These external forces are in fact essential. It is not possible to walk without interacting with the floor and the friction and reaction forces generated when the feet collide with the floor. Certainly a more automatic method of deriving such force or torque functions is essential

2.6.2 Motion Controllers

In the last few years the notion of motion controllers has been introduced for parameterizing force or torque functions in a more compact and convenient manner. A motion controller provides a higher level description of motion than force or torque functions. (cf Figure 2.7). The task of the motion controller

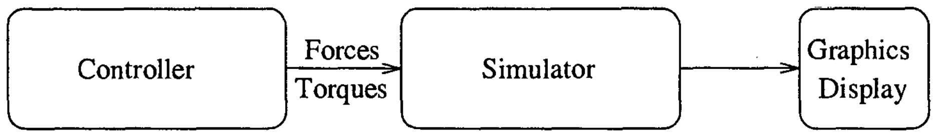


Figure 2.7: Function of a controller

is to produce the necessary forces or torques that in turn will generate the desired motion. A variety of different representations have been evolved for these motion controllers and we shall describe them shortly. All motion controllers, however are essentially finite state machines, which have to be executed in order to generate force or torque functions of time. Thus the problem of specifying a continuous force or torque function is transformed to that of specifying static states and state transitions rules for the motion controller. Apart from providing a compact and static parameterization of the force or torque functions, motion controllers have the additional advantage that smoothness in the variation of force or torque can also be built in.

This is done by associating actuators with the joints of the articulated figure. Actuators convert stored energy into time varying forces or torques. Typical example of actuators are springs and muscles. The actuator modelled as a spring and damper system is one of the simplest compliant actuator. It consists of a spring and damper mechanism in which a torsional spring attached between two links applies a torque on the adjacent links according to the *proportional derivative* (PD) control law:

$$\tau = k_p(\theta_d - \theta) - k_v\,\dot{\theta}$$

where k_p is the spring constant, k_v is the damper constant, θ and $\dot{\theta}$ are the current angle and the angular velocity respectively and θ_d is the rest angle (equilibrium position) of the spring. (cf figure 2.8) The values of k_p and k_v

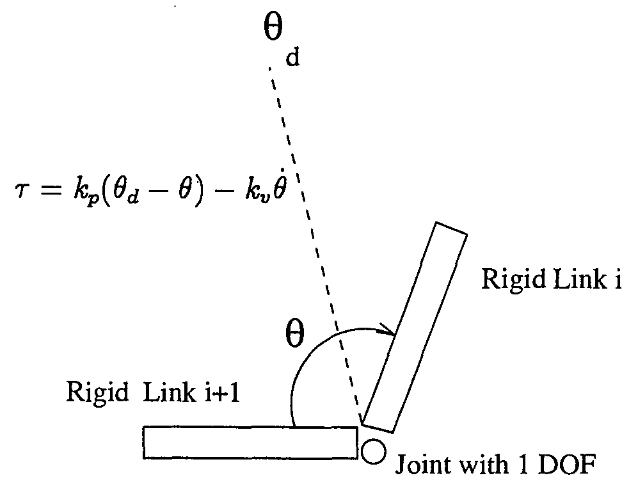


Figure 2.8: Actuator modelling spring and damper mechanism

are typically chosen such that the mechanism is critically damped. A critically damped system is one that when disturbed will most rapidly return to equilibrium position. Such a system acts to control the angular position, in that, if

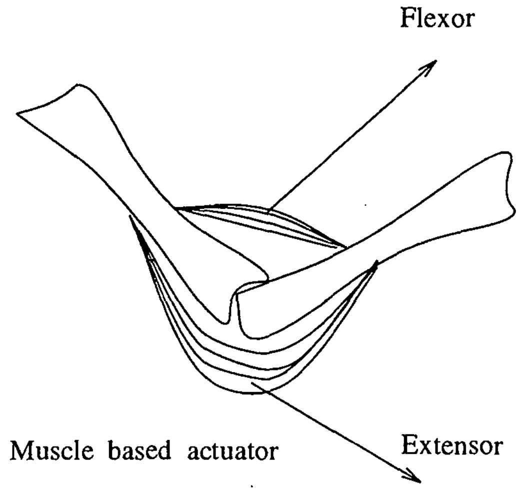


Figure 2.9: Biomechanical actuator

the current angle is different from the equilibrium angle, the spring will apply a torque on the adjacent links so as to again be in equilibrium. For example, if the current limb joint angle is less than its desired angle, the mechanism will cause a positive torque to be applied to the joint to move it back toward the desired angle and vice versa. The velocity damping term reduces the torque applied to the joint once movement towards the desired angle is underway. By changing the desired angle at different instances of time, the mechanism can be actuated to bring about the motion.

Lately, researchers in computer animation are looking at biomechanical models to generate more realistic motion [47]. These methods build from dynamic physical simulation by concentrating on the motion of living, biological system that obey biomechanical principles. In particular, biomechanical methods develop and use various models for the muscle and tendon dynamics that act as biological force actuators in these physical systems. Victor Ng [47] replaces the motor actuators with a muscle pair known as flexor and extensor (cf figure 2.9). These two muscles work together in synergy to create motion. For example, holding an arm straight out involves alternating work by the flexor and the extensor in the arm to adjust the arm level.

The general problem of motion control with controllers is to devise schemes for activating the actuators to produce coordinated movement. There are a number of different motion controller representations that have been proposed. These are briefly described below:

Finite State Machine Controllers

One of the most popular representations for a controller is using the finite state machine (FSM). Typically such a controller consists of a set of states and a set of rules, which together specify the torques to be applied at the joints to bring about the motion. Zeltzer [71] was one of the first researchers to create walking FSM controllers by dividing up the various phases of the walking gait into low level motor programs. Raibert and Hodgins [87] have used FSM controllers to model hopping, speed control and posture control. They treat each state as an active control law for various phases of motion. A PD control is used to move an articulated figure to various calculated positions based on the current state. Stewart and Cremer [95] have developed algorithms to control a biped climbing and descending stairs by dividing the motion into phases such as double-support, start-swing-up and swing etc. Recently, Hansen *et al.* [40] have proposed motion control through communicating hierarchical state machines combined with constraint based control to specify control strategies for tasks such as hopping, walking, balancing etc.

In an FSM controller, each node of the FSM represents a state and each arc an action to be performed by the articulated figure. Typically a state is entered when a certain event in the articulated figure is sensed. For example, a leg leaves the ground or touches the ground etc. Actions are at a fairly high level and each action would need a specifically programmed procedure for generating the torque. For example, interchange active idle legs, lengthen active leg for landing etc. Figure 2.10 and associated Table 2.1 show the FSM controller designed by Raibert and Hodgins [87] for simulating the running movement of a biped. As such in this approach both states and actions are very specific to the desired movement.

Thus for each single movement, say walk, jump, hop etc. there has to be a

State	Action
FLIGHT	
Active leg leaves ground	Interchange active, idle legs Lengthen active leg for landing Position active leg for landing Shorten idle leg
LOADING	
Active leg touches ground	Zero active hip torque Keep idle leg short
COMPRESSION	
Active leg spring shorten	Extend active leg Keep idle leg short
THRUST	
Active leg spring lengthens	Extend active leg Servo pitch with active hip Keep idle leg short
UNLOADING	
Active leg spring approaches full length	Shorten active leg Zero hip torques active leg Keep idle leg short

Table 2.1: An FSM that coordinates running for a biped

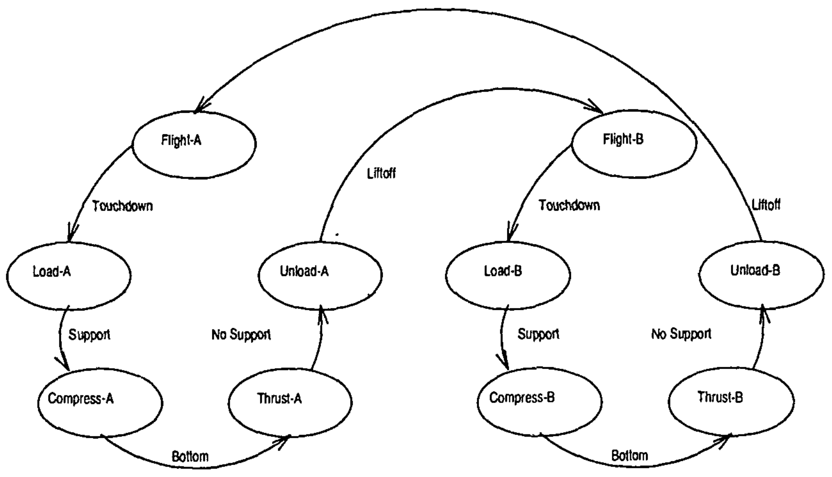


Figure 2.10: Controller for a two legged articulated figure

separate FSM controller designed with different topologies and also different states and actions. The method is certainly not generic enough for automated synthesis of controllers. To date, these types of controllers have all been hand designed and tuned.

Pose Control Graphs

Pose control graph is another representation for a motion controller proposed by Van de Panne *et al.* [103]. This provides only open loop control but in a generic fashion. Each pose control graph is either cyclic or acyclic and consists of a number of nodes, where with each node is associated a pose of the articulated figure. A pose is a static posture of the articulated figure and is basically specified by fixing all the joint angles. Associated with every arc is a time period. (cf Figure 2.11) Transition from one state to another is completely determined by the time period associated with the transition arc. The pose in the end node of this arc provides a kind of "goal" that the articulated figure has to reach. This is done through the use of spring and damper actuators associated with each joint. Independent of the current posture of the articulated figure, the pose control graph tries to reach the destination posture as dictated by the damping coefficients associated with each joint and the torque to be applied is

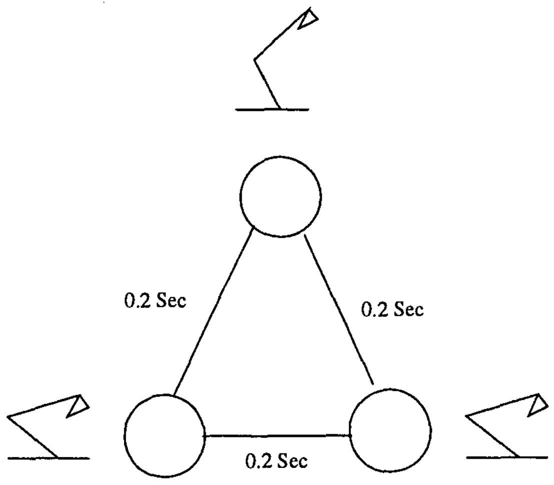


Figure 2.11: Pose control graph for a hopping lamp

determined by the distance of the spring from the rest position. Rest position is the joint angle in the end node pose. As soon as the transition time is completed the pose control graph controller will switch to the next state even if the articulated figure has not achieved the goal. A new transition now starts. A few important observations are as follows:

- Poses are like key-frames, except that they are used more to derive the torque functions with the help of spring and damper actuators.
- Articulated figures moving with the help of pose control graph based motion controllers are like wind-up toys. The force and torque functions get applied independent of the environment or the interaction with the environment. Thus a wind-up toy will flap its legs and move when placed on a floor, but will just flap its legs if held in the air or if obstructed by a wall. Since their execution is totally determined by the transition times and does not depend in any way on the sensing of the figure's state or environment dependent events, we refer to these as open loop controllers.
- Because of their open loop nature, pose control graph controllers can exhibit only a limited class of motion behaviours. But this is not necessarily trivial. In fact complex motion like walking, hopping, jumping etc. can all be synthesized using pose control graphs. Pose control graph as a

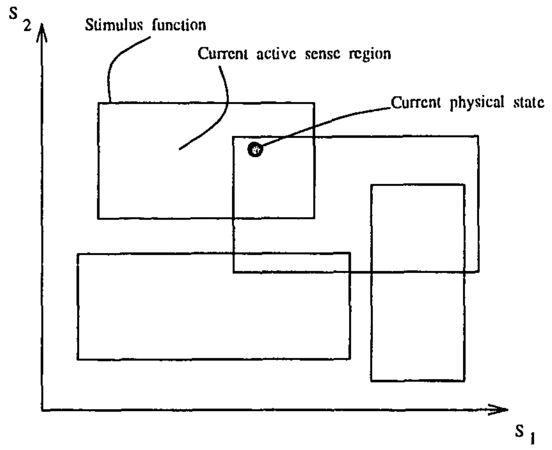


Figure 2.12: Banked stimulus response controller with two sense variables

representation for motion controllers is simple and generic enough for experimenting with automated synthesis.

Banked Stimulus Response

Stimulus response is a powerful paradigm that tries to mimic the behaviour of living creatures. A controller based on the stimulus response paradigm makes use of the knowledge of the external environment to drive its motion. Banked Stimulus Response (BSR) and Sensor Actuator Networks (SAN) are the two controllers which are based on this paradigm.

The principle concepts underlying BSR representation are sense variables, stimulus functions and associated action rules. A sense variable is some real-valued function of the object's physical state. Every physical state of the object is mapped onto a point in the sense space. A stimulus function is a scalar function defined over sense space that is negative everywhere except over a small region, which is called sense region. (cf Figure 2.12.) Associated with each stimulus function is a response, which prescribes some action for the object. As the physical state of the object changes, the corresponding point in the sense space moves from one sense region to the other. Every time it changes the sense region the corresponding action associated with the particular stimulus

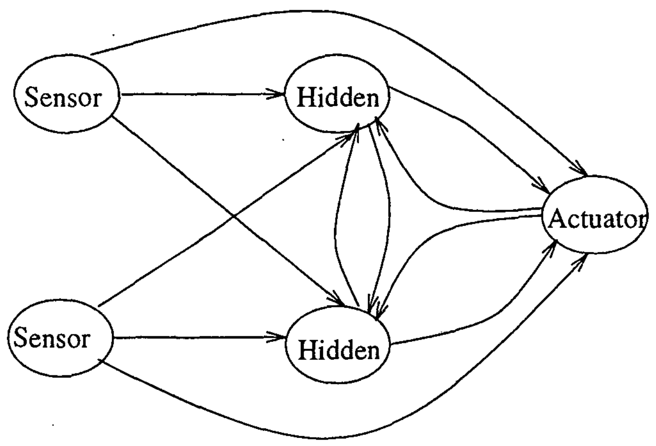


Figure 2.13: Topology of SAN showing different nodes

function is applied and the process continues. The mechanism makes sure that at all times there will only be a single sense region which will be active and the action(s) corresponding to that region is performed.

Sensor Actuator Network (SAN)

The SAN controllers suggested by Van de Panne *et al.* [100] provide control by connecting sensors to actuators in the form of a network of weighted connections. All sensors are defined to be binary. That is, if the sensor is on, it produces a value of 1 otherwise it produces a value of 0. A typical example of a sensor would be a touch sensor that would turn on when in contact with the ground and otherwise remain off. Once again actuators are modelled using PD control law. An example of a SAN is shown in the Figure 2.13. The network consists of nodes and unidirectional weighted connections. Once the weights are synthesized, the SAN maps the sensor information to action through actuators to bring about a variety of motions. In its function, a SAN is very similar to an artificial neural network.

Some Problems in Controller Specification

While the parameterization of motion controllers is done so that user specification for desired motion is simpler than direct specification of force and torque functions, it still does not provide a natural and intuitive method for the animators. Most of the experience in synthesizing motion using motion controllers has shown the following:

- The choice of the topology for the motion controller for a specific movement is a difficult one. There are no studies as yet that provide any definite guidelines towards this.
- The definition of states and associated transition parameters has to be carefully hand tuned. Considerable experimentation is necessary before the desired motion controller is obtained.
- All the motion controllers that have produced natural looking simulations described in the literature are results of extensive studies of captured motion from live actors performing similar movements. A deep understanding of the real life motion and also the controller behaviour is essential for anyone to design a suitable motion controller.

2.7 Automatic Motion Synthesis

Taking all the above into account a large number of researchers have pursued automated synthesis of force and torque functions and also motion controllers [100, 102, 81]. Like inverse kinematics these methods could all be classified as being in the general category of inverse dynamics. Once again these inverse methods are solved using constrained optimization techniques.

2.7.1 The Constrained Optimization Problem

The constrained optimization problem is formally described below:
Let a system be characterized by n state variables $x_1, x_2, \ldots, x_n$ and written as

the state vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. Further, assume that the state variables satisfy the coupled first-order differential equations

$$\frac{dx_i}{dt} = f_i(x_1, x_2, \ldots, x_n; u_1, u_2, \ldots, u_m; t) \quad 1 \leq i \leq n$$

on $[0, T]$ where m variables $u_1, u_2, \ldots, u_m$ form the control vector $\mathbf{u} = (u_1, u_2, \ldots, u_m)$. The motion control problem can then be formally stated as:

> Find an admissible control $\mathbf{u}^*$ which causes the system
>
> $$\dot{\mathbf{x}}\,(t) = \mathbf{a}(\mathbf{x}(t), \mathbf{u}(t), t)$$
>
> to follow an admissible trajectory $\mathbf{x}^*$ that minimizes the performance measure
>
> $$J = h(\mathbf{x}(T), T) + \int_0^T g(\mathbf{x}(t), \mathbf{u}(t), t)dt$$
>
> where $\mathbf{u}^*$ is called optimal control and $\mathbf{x}^*$ is called optimal trajectory.

Here h and g are scalar functions and by admissible control we mean control variables and state variables satisfying all the control constraints and state variable constraints over the entire interval $[0, T]$. Let U represent admissible control space and X represent admissible state variable space.

Starting from the initial state $\mathbf{x}(0) = \mathbf{x}_0$ and applying different control signals $\mathbf{u}(t)$, over the interval $[0, T]$; the system will generate various state trajectories. The performance measure assigns a unique real number to each of these trajectories. When we say $\mathbf{u}^*$ causes the performance measure to be minimized we mean that for all $\mathbf{u}\epsilon\mathrm{U}$, which make $\mathbf{x}\epsilon\mathrm{X}$, the performance measure is smaller than any other admissible control. In other words we are seeking the absolute or global minima of J and not local minima (cf Figure 2.14). It is important to note here that many a times it is not possible to know in advance whether there exists any optimal control $\mathbf{u}^*$. Also, even if optimal control exists it may not be unique.

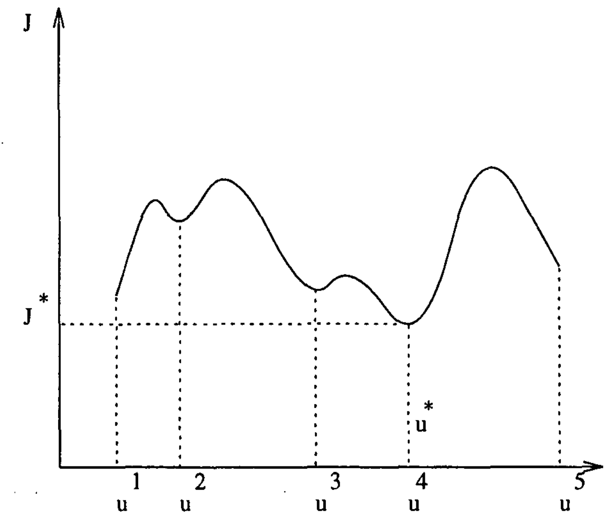


Figure 2.14: Performance metric J for different control signals $u(t)$

2.7.2 Dynamic Constraints

As we have seen earlier, in dynamic methods, all interaction among bodies and between a body and its environment are mediated through forces and torques. In particular, if we wish to influence the behaviour of a particular articulated figure, we must do so through application of forces. The "dynamic constraints" method described by Barzel and Barr [11] uses inverse dynamics to determine the forces which influence the behaviour of the bodies. However, to express the motion behaviour of an active articulated figure through constraints is non-trivial. A similar idea was put forward by Issac and Cohen [52]. Typically the user specifies a desired behaviour through a set of constraints such as "point-to-point", "point-to-nail" etc. The system then determines the unknown forces needed in order to meet the constraint. The equation to be solved is linear in forces and torques. The method is quite effective in specifying the motion of passive articulated bodies such as pendulums and chains swinging under the influence of gravity and other user specified forces and torques. Once the constraint forces are known, they are added to the simulator canceling exactly the components of the applied forces that fight against the constraint. However, specifying dynamic constraints for articulated figures to bring about natural looking motion is non-trivial.

2.7.3 Space-Time Constraints

In this method constraints are in state space as well as in time. Basically these methods attempt to find a trajectory in state space that optimizes a performance metric J. J provides a quantitative measure for desired motion qualities of any trajectory. Clearly the formulation of this performance metric J will influence the trajectory shape and consequently the quality of the motion.

The spacetime constraint method was introduced simultaneously by Witkin and Kass [108] and by Brotman and Netravali [15]. Both use key poses (key-frames) fixed in time as constraints for trajectory optimization. Brotman and Netravali give a method of obtaining an optimal trajectory that interpolates the given key-frame constraints with a performance metric that ensures smooth trajectories and minimal control energy. Witkin and Kass only minimize control energy and do not necessarily interpolate the key-frames. The principal difference is as follows. Brotman and Netravali determine a piecewise trajectory, each piece being between two key-frames and with smoothness ensured at the junctions of two connected trajectory pieces. On other hand Witkin and Kass consider all the key-frames together and find the complete trajectory. Thus in the first method by Brotman and Netravali, J is approximated by a vector with components equal in number to the intervals between successive key-frames. A multi-point boundary value problem is converted to a series of two point boundary value problems. In the Witkin and Kass method, they choose to include the dynamic equations of motion as constraints, it results in a large system of equations that do not necessarily converge. In both cases the dynamic equations of motion are included in such a fashion so that the optimal trajectory would be encouraged to satisfy these equations but may not always satisfy exactly. Both solution methods finally result in the form of force and torque functions to be applied at each of the joints. Since the force or torque function is independent of the state x and $\dot{x}$, this is a type of open loop control where no state feedback can influence the control forces or torques. Both methods use local optimization techniques that find a solution trajectory that is as close to physical realizability as possible.

In the spacetime constraints method optimization is in time discretized state variables. Along with many constraints this large number of state variables make this method slow and impractical. Two extensions have been evolved to overcome this problem.

1. Spacetime windows are separate subdivided regions of spacetime variable space enabling piecemeal building up of trajectory like that by Brotman and Netravali. Except here the spacetime windows are not necessarily bounded by successive key-frames. The larger optimization problem is decomposed into series of smaller problems, each of which can converge quickly. At the same time an animator is provided with greater control over the animation by means of more windows, constraints and goals.

2. Parameterized trajectories are more compact representations as they resort to the use of a higher level functional basis like linear, β-spline or wavelets. Instead of trajectories represented in a finely discretized fashion, trajectories are replaced by piecewise continuous functions with far fewer parameters. An undesirable side effect due to the use of basis functions like β-splines is that the resulting trajectory may end up with artificial smoothness and excess control energy. Secondly, these piecewise functions require a minimum number of key-frames within each spacetime window. As the number of spacetime windows increases the method degenerates to the traditional key-frame interpolation with all associated disadvantages as well.

2.7.4 State Space Motion Controllers

One of the earliest methods of automatically synthesizing a motion controller is the state space controller method [101]. A state space controller generates a set of control torques that guides an articulated figure to a specified end state from some given initial configuration while satisfying all the given constraints and also optimizing a stated goal. The state space controller is represented as a dynamic programming graph in discretized state space of the articulated figure. The graph defines optimal torque values to be applied to the joints of the articulated figure for reaching a specified destination state. Motions are optimized with respect to time and control energy. The principle of dynamic programming which states that all intermediate paths are also optimal, ensures that any path existing between two nodes in the graph also represents an optimal path. Thus instead of the previous two point boundary value optimization, a state space controller represents a family of optimal solutions from many different initial states to a single common destination state. This makes the state space controller reusable, albeit in a limited sense.

For each distinct destination state, a different dynamic programming controller graph has to be computed. Each controller is defined over a bounded domain of the state space. Provided these domains overlap and in the optimal paths there exists a common state, it is possible to concatenate controllers and produce complex motion. While the dynamic programming search technique is global, the state space controller method essentially results in local optimization, because each dynamic programming graph controller is optimal only over the restricted region of the state space, that is the domain of the controller. Another problem that is inherent due to the restricted state space domain of the controllers is that unanticipated interactions with the environment like collisions cannot be handled as the post collision state may not be covered by any of the predefined set of controllers. The state space controller method is significant primarily because of the fact that it was one of the first attempts at automated controller synthesis. However, the need for fine discretization of the state space makes the method largely impractical.

2.7.5 Automated Motion Controller Synthesis

With the tremendous rate at which computing performance improvements take place, it was but natural that researchers would explore global optimization techniques in place of the local search. Optimization techniques for parameterized controller-based spaces are relatively of much less dimension than state space. Current research is primarily directed towards this. In fact automatic synthesis is the primary motivation for introducing a variety of motion controllers such as the pose control graph [103], the banked stimulus response [81] and the sensor actuator networks [100].

In global optimization techniques, the initial close guess that is needed in local optimization methods is not necessary. The search for controllers expands beyond local regions and covers the entire global domain of the controller parameter spaces, where several possible solutions can be found, each with potentially disparate motion characteristics. All the global optimization techniques proposed for obtaining optimized controllers are based on some form of genetic programming. These genetic procedures have been used to synthesize all the different types of controllers. Basically in genetic programming, an initial population of controllers is chosen randomly. Subsequent generations of the controller population are obtained by refinement that uses operations like mate selection, cross over and mutation. Over a sufficiently large number of generations an optimal controller (not guaranteed to be global though) would be synthesized.

The application of global optimization techniques to controller synthesis is certainly the most promising approach to physically based animation today. While the available results are very impressive and seem to hold out the promise, currently available techniques are far from what an animator would like. The prime problem is one of having fine control over the resulting motion. Without this control producing animation sequences that follow a script will be extremely difficult and almost not possible. The only method currently available for controlling the synthesized motion is through the use of the fitness function. Fitness functions for specific articulated figure configurations for specific movements like walk, hop, jump etc. have been evolved. There is however no direct or intuitive association between the fitness function and the final desired motion.

2.8 Important Issues in Articulated Figure Animation

From the above descriptions, it is clear that automated synthesis of parameterized motion controllers using global optimization techniques has the potential to be a powerful computer graphics tool for animators to create animations involving complex movements of virtual creatures movements that are physically correct, realistic and natural looking. There are three aspects of the problem that are very important in using such methods.

1. **Search Technique**
   The solution space of parameterized controllers is not only of high dimensionality, but also is multi modal and not always continuous. Gradient based numerical methods like steepest descent cannot always be used. Global methods like genetic algorithms have to be employed. Certainly characteristics of the search space and choice of a suitable search method would be crucial to the resulting efficiency and success of the automated motion controller synthesis technique. Chapter 3 includes a detailed description of optimal search methods and their applicability.

2. **Environmental Interaction**
   Active articulated figures when moved in a virtual environment, consisting of other virtual objects or figures are bound to collide with them. Any physically realistic movement simulation would not only have to take care that virtual objects do not move through each other on collision, but in fact also react in a physically correct manner. While closed loop controllers do have the potential to accommodate this reaction within their state-action frame work, most of the existing methods deal with collision and response through a separate simulation module. This is discussed in greater detail in chapter 4.

3. **Animator Control**
   Certainly all possible motions are not necessarily optimal in terms of global properties like speed or control energy. Thus control of the automatic motion controller synthesis process such that the resulting motion is as desired by the animator is a problem that needs to be attended to. Not only must the animator have good physical intuition but he or she must also have a good feel for the synthesis process so that a suitable performance metric can be designed. In fact there are probably several

goals to be concurrently optimized and possibly in a conflicting manner. Chapter 5 presents complete details of our solution to this problem using motion features and Chapter 6 describes the implementation of a system that automatically synthesizes motion controllers based on desirable features specified by an animator. Chapter 6 also includes results from some of the experiments conducted using this implementation for animating different types of movements of a few virtual creatures.

Chapter 3

Optimization Techniques for Motion Synthesis

As discussed in chapter 2 the most popular and promising approach to the problem of motion synthesis for active articulated figures is through the use of constrained optimization techniques. There are five components of the constrained optimization problem. These are as follows:

1. The search space that includes all solutions.
2. A task goal defined mathematically via a performance metric.
3. A dynamic system that is to be controlled.
4. A set of constraints.
5. An analytical or numerical algorithm capable of finding an optimal solution.

In this chapter we discuss how the different motion synthesis methods described earlier handle each of the components.

3.1 The Search Space

The size and dimensionality of the search space in which the optimal solution has to be found has a profound impact on the overall performance of the method. Usually constraints are used to limit the search space and these are discussed later in this chapter. The dimensionality is determined by the number of parameters needed to uniquely specify a single movement (solution). As the number of DoFs increase, the dimensionality of the search space increases.

All the early methods like key-frame animation and inverse kinematics searched directly in trajectory space, the space formed by position variables and time. The parameters for uniquely specifying a solution essentially defined a path as position varying with time. On the other hand, all the initial physically based animation techniques carried out the search in state variables (that is DoF and first derivative ~~of each DoF with~~ respect to time). For example the methods by Witkin and Kass [108] and Brotman and Netravali [15] simply used state variables discretized in time as parameters. There are obviously a number of severe problems with this.

- First and foremost, as the actual time duration of the movement is increased the dimensionality of the search space increases.
- Secondly, as the time interval used for discretization is reduced it once again results in increased dimensionality. While we need really only 24 (or 30) frames per second for final playback, often the nature of differential equations of motion that are involved is such that much finer steps have to be considered.
- Lastly, as we shall see in the next chapter, external interactions like collisions and their resolution also demand that the time intervals be carefully chosen. This again increases the search space dimensionality.

Large optimization problems are slow to converge. A number of efforts have been made to essentially address this problem. Cohen [20] proposed subdividing the animation into smaller pieces. This decomposes the larger optimization

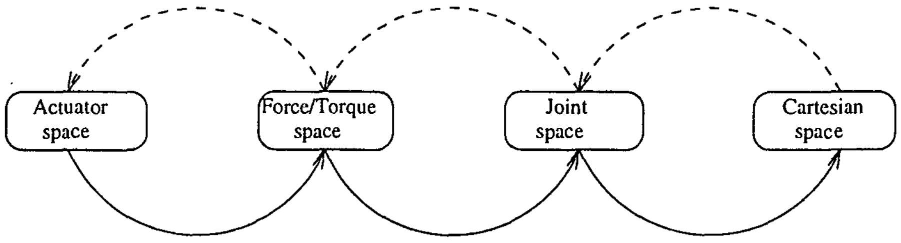


Figure 3.1: Mappings between different coordinate spaces (dotted line shows inverse mappings)

problem into a series of smaller problems which can give the animator greater control over the final animation by creating more windows and constraints. However as the number of windows increases the animator's efforts also increase considerably and the computer support effectively reduces to that of providing interpolation facilities. Similarly the formulations ~~dealing with pa-~~ rameterizing trajectories using β-splines or wavelets also attempt to reduce the dimensionality of the search space. Instead of a large number of discrete points representing a continuous trajectory, a cubic β-spline for example, would require only 4 control points in space-time to represent that continuous trajectory. Similarly with wavelets using only a few significant components the continuous trajectory can be represented. As already mentioned the main disadvantage is the forcing of higher order continuity in the trajectories that is inherent in these methods, thus making these methods suitable only for motion with smooth trajectories.

All subsequently proposed methods have been based on the strategy of finding the optimal motion in a space different from the state space. As seen in the previous chapter, this is done by introducing the notion of motion controllers that are responsible for generating the necessary forces/torques for a desired motion. Figure 3.1 shows the different search spaces being used in animation.

1. Most of the kinematic methods, particularly key-frame animation techniques, use Cartesian space (2D or 3D).

2. Joint space, denoted by joint angles of different links of the articulated figure is used by inverse kinematic methods. It may be recalled that

transforming from joint space to Cartesian space is easily done using forward kinematics.

3. Force/torque space or control space is used by most of the initial dynamics based methods. Specifically methods like the dynamic constraints [11], space-time constraints [108] and its extension [20], and also the state space controller method based on dynamic programming [101], all search for the optimal trajectory in control space. Once the trajectory is found in control space, then by formulating and solving the dynamic equations of motion, it can be transformed to joint space.

4. Actuator space depends on the type of actuators modelled in the motion controller. For example if the spring and damper actuator model is used then the space is defined by spring constants and rest lengths/angles. All the newly proposed automatic motion controller synthesis techniques find the optimal trajectory in actuator space. There are ongoing studies to use mathematical muscle models from biomechanical studies as actuator models [47]. These offer the potential to produce alternative motions through the use of performance metrics that accurately represent features that are being optimized in real motions. Certainly biologically correct muscle models can help in matching and evaluating simulated motion with digitally captured live motions.

While actuator space is definitely of lesser dimension than trajectory or torque space, the search space dimensionality continues to be a major hurdle in the use of these techniques. Ng refers to this as the curse of dimensionality [48]. Efforts will certainly continue towards defining control methods that will further reduce the search space dimensionality.

3.2 The Performance Metric

With the motion synthesis problem transformed to one of non-linear constrained optimization, an animator can control the resulting movements only through the specification of the performance metric, (objective function that is to be optimized) and the constraints that the motion has to satisfy. As stated earlier, the performance metric is usually a functional of the form

$$J = h(\mathrm{x}(T), T) + \int_0^T g(\mathrm{x}(t), \mathbf{u}(t), t)dt \tag{3.1}$$

where g is integrated over the time duration of the animated sequence T and h is a scalar function that is evaluated only at the end of the time duration T. Without loss of generality we can assume that J is minimal for the optimal motion.

A variety of performance metrics have been suggested and used. The space time constraints method usually minimizes energy, that is J takes the form:

$$J = \int_0^T |u(t)|^2 \, dt \tag{3.2}$$

Though there have been indications that other formulations of the performance metric could also be used in these methods, in the published literature so far we do not see any example of other performance metrics used with the space-time constraints methods.

In the state-space controller [101] methods both time and energy are used by suitably weighting each. Thus J takes the form

$$J = \alpha T + (1 - \alpha) \int_0^T |\mathbf{u}(t)|^2 \, dt \tag{3.3}$$

There are other variations of the metric formulations:

1. The distance traveled by the creature with a penalty for moving backward [102].
$$J = -x(T) + \max_{0 \leq t \leq T} x(t)$$
Here x represents the distance travelled.

2. Reach a specified pose without loss of contact with the ground (falling over) [80].
$$J = \Sigma_{i=1}^{N} |\theta_i(T) - \theta_i^0| + \max_{0 \leq t \leq T} y(t)$$
Here θ_i^0 represents the desired configuration of the i^{th} link and $y(t)$ the lowest point on the body.

3. Jump to achieve a maximum height. J is maximized such that lowest point on the body $y(t)$ is reaching the maximum height [80].

$$J = \max_{0 \leq t \leq T} y(t)$$

4. To achieve energetic hops, one can use

$$J = \int_0^T (\dot{x} + ky^2) dt$$

Here $\dot{x}\ dt$ measures the the distance travelled and $ky^2 dt$ measures the average height of the creature [100].

5. The distance travelled in a fixed amount of time [103].

$$J = |x(T)|$$

Here the assumption is that motions that end farthest are usually the ones that have an interesting mode of locomotion.

6. To follow a moving target,

$$J = \int_0^T \dot{p}\ .F dt$$

Here p is the position of the creature, $\dot{p}$ is its velocity and F the unit vector pointing towards the target [100].

While the emphasis in most of the research carried out so far has been on reducing the search space dimensionality and also on effective optimal search techniques, it is clear that we have to develop simple methods that provide good physical intuition to derive performance metrics that result in desired motions. And metrics that permit the concurrent optimization of several goals, thus raising further the problem of distributing the weights amongst individual goals that may even be in conflict.

The early efforts were to subdivide the animation and carry out the motion synthesis task piecewise. Each piece could in principle be optimizing a different performance metric. Both Brotman and Netravali and the spacetime windows based methods are examples of this. Restricting the state-space region over which the dynamic programming based state space controller is derived, coupled with the concatenation of state space controllers for more complex motions is also another case of a similar approach. As a specific case Cohen [20] argues that space time windows can simulate a cat chasing a mouse by giving the cat only local knowledge of the state space around it. In contrast, global knowledge would feature the cat moving in a straight path directly to meet the mouse at the destination point. However, a very large number of windows had to be created to generate a tight chasing scene. This example, notwithstanding, a major draw back of these methods is that they make the optimization essentially local by restricting the domain of the search. And such local optimal solutions may actually be far from the desired motion in most cases.

A slightly more generalized method for formulating a performance metric has been described in [37]. In this approach the metric is formulated in the form of a main goal and style points. The main goal is typically a simple metric of whether the primary requirement of desired motion has been fulfilled. For example, if the goal of the motion sequence is to move the figure to the point "X", then the main goal would simply be the distance between the figure and "X" at the end of the time allotment (it must be noted that lower numbers correspond to better performance).

Since the motion is mostly underconstrained, the system can often find outrageous ways of satisfying such a simple performance metric. For example, it might somersault to the goal point instead of hopping. Because of this, style points are added which can be thought of as additional rewards or penalties granted to the virtual creature's performance. Examples of style points include:

1. Penalties for hitting obstacles or violating safety rules (don't hit your head on the floor).
2. Rewards for performing the action quickly, or slowly.
3. Penalties for inefficient behavior (such as taking the long way around an obstacle or sitting for a long time, then rushing when it gets close to the

time limit).

4. Rewards for ending in "neutral positions" and remaining in control (you do not want the creature tangled up or laid across the floor when it completes its action).

5. Rewards for minimizing energy consumption are also useful at times. Interestingly however, most of the time, they found that this did not improve the quality of motion in any significant way. Even when energy considerations were useful, the effects tended to be very subtle, and not as important as the other style considerations.

6. Problem-dependent terms (for example, whether certain subgoals were met).

A primary draw back of this method is that the performance metric has to be programmed separately for each motion. The style points are very specific to the virtual creature and the type of movement, and demand programming capabilities, say, at the 'C' language level, from the animator and/or associates. Certainly a simpler, more generic and declarative type of method has to be evolved.

3.3 The Dynamic System

Given the mass of the object, its moment of inertia, forces and torques acting on the object and the constraints to be satisfied at any given point of time, the dynamic system is formed as a system of differential equations. These equations known as the dynamic equations of motion basically relate how the mass moves under the influence of forces and torques. Resolving these equations for acceleration and then integrating the equations enables us to obtain new velocities and positions. These new positions are used to provide the desired animation. The description of the motion can be entirely in one, two or three dimensions. Below we describe in detail the format for the physical description of objects, methods for formulating the dynamic equations of motion, solution methods for resolving the accelerations and finally integration techniques to obtain the positions and velocities. We also critically address issues such as stability, stiffness and computational efficiency relating to the above.

3.3.1 The Physical Object Description

In the simplest case, the object is represented as a point mass, and the only physical description needed is its mass. More realism is obtained by treating the object as a rigid body, which is made up of masses distributed in either 2D or 3D space. The information needed to simulate a rigid body includes its total mass, centre of mass, and moment of inertia. The moment of inertia of a rigid body may be defined relative to any frame, either attached to the body or outside. The inertia tensor relative to a frame A is expressed in matrix form as the 3×3 matrix:

$$ {}^{A}I = \begin{pmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{pmatrix} \tag{3.4} $$

where the scalar elements are given by:

$$ \begin{aligned} I_{xx} &= \int\int\int_V (y^2 + z^2)\rho dv \\ I_{yy} &= \int\int\int_V (x^2 + z^2)\rho dv \\ I_{zz} &= \int\int\int_V (x^2 + y^2)\rho dv \\ I_{xy} &= \int\int\int_V xy\ \rho\ dv \\ I_{xz} &= \int\int\int_V xz\ \rho\ dv \\ I_{yz} &= \int\int\int_V yz\ \rho\ dv \end{aligned} $$

and the rigid body is composed of different volume elements, dv, containing material of density ρ.

The elements I_{xx}, I_{yy}, and I_{zz} are called the *mass moment of inertia*. The elements with mixed indices are called the *mass products of inertia*. If the reference frame is attached to the centre of mass of the body, the products of inertia terms vanish and the resulting inertia tensor can be written as:

$$^{C}I = \begin{pmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{pmatrix} \tag{3.5}$$

For a simple rectangular box of homogeneous material with dimension $l \times w \times h$ inertia tensor with respect to the frame attached at the centre of mass will look like:

$$^{C}I = \begin{pmatrix} \frac{m}{12}(h^2+l^2) & 0 & 0 \\ 0 & \frac{m}{12}(w^2+h^2) & 0 \\ 0 & 0 & \frac{m}{12}(l^2+w^2) \end{pmatrix} \tag{3.6}$$

3.3.2 Dynamic Equations of Motion for Articulated Figure

There are number of ways to formulate the dynamic equations for an articulated figure. In computer animation, researchers have used Gibbs-Appell [105], Lagrangian [4], D'Alemberts [52], Armstrong's method [3] and Newton-Euler formulation [23]. All the approaches are equivalent and yield the same results but the route to the solution in each case is different. The key issue lies in the computational efficiency of the solution. In this respect it is well known that recursive formulations are far superior when compared to their non-recursive counterparts. We now discuss, the most popular and also highly intuitive method, the Newton-Euler method.

For a single rigid body, Newton-Euler equations can be written as two three-dimensional vector equations, a Newton's equation that typically gives the linear motion of the centre of mass, and an Euler's equation that gives the rotational motion.

Newton's Equation

Consider a rigid body whose center of mass is accelerating with acceleration $\dot{v}_c\ (t)$. In such a situation, the force $F(t)$ acting at the center of mass which causes the acceleration is given by Newton's Equation

$$m\,\dot{v}_c\,(t) = \boldsymbol{F}(t) \tag{3.7}$$

Euler's Equation

Consider a rigid body rotating with angular velocity, ω, and with angular acceleration $\dot{\omega}$. In such a situation, the torque $\boldsymbol{N}(t)$, which must be acting on the body to cause the rotational motion, is given by Euler's equation

$${}^{C}I\,\dot{\omega}\,(t) + \omega(t) \times {}^{C}I\omega(t) = \boldsymbol{N}(t) \tag{3.8}$$

These are 2nd order differential equations which need to be solved at each time step in order to get the position and orientation of body.

However, for an articulated body made of rigid links, the movement of the links is not completely free but is constrained by the joints. In fact, neighbouring links exert forces on each other, which restrict the relative motion between the links. For example, if we consider only rotary joints, there exist only 3 DoFs between the joints and no translational motion will be allowed between adjacent links.

The complete algorithm for forming the equations of motion is composed of two parts. First, link velocities and accelerations are iteratively formulated from link 1 out to link n and the Newton-Euler equations are applied to each link. Second, forces and joint torques are computed recursively from link n back to link 1.

The equations are as follows:

Outward iterations: $i : 0 \rightarrow n-1$

$$\begin{aligned}
{}^{i+1}\omega_{i+1} &= {}^{i+1}_{i}R\,{}^{i}\omega_i + \dot{\theta}_{i+1}\,{}^{i+1}\hat{Z}_{i+1}, \\
{}^{i+1}\dot{\omega}_{i+1} &= {}^{i+1}_{i}R\,{}^{i}\dot{\omega}_i + {}^{i+1}_{i}R\,{}^{i}\omega_i \times \dot{\theta}_{i+1}\,{}^{i+1}\hat{Z}_{i+1} + \ddot{\theta}_{i+1}\,{}^{i+1}\hat{Z}_{i+1}, \\
{}^{i+1}\dot{v}_{i+1} &= {}^{i+1}_{i}R\left({}^{i}\dot{\omega}_i \times {}^{i}P_{i+1} + {}^{i}\omega_i \times \left({}^{i}\omega_i \times {}^{i}P_{i+1}\right) + {}^{i}\dot{v}_i\right), \\
{}^{i+1}\dot{v}_{C_{i+1}} &= {}^{i+1}\dot{\omega}_{i+1} \times {}^{i+1}P_{C_{i+1}} + {}^{i+1}\omega_{i+1} \times \left({}^{i+1}\omega_{i+1} \times {}^{i+1}P_{C_{i+1}}\right) + {}^{i+1}\dot{v}_{i+1}, \\
{}^{i+1}F_{i+1} &= m_{i+1}\,{}^{i+1}\dot{v}_{C_{i+1}}, \\
{}^{i+1}N_{i+1} &= {}^{C_{i+1}}I_{i+1}\,{}^{i+1}\dot{\omega}_{i+1} + {}^{i+1}\omega_{i+1} \times {}^{C_{i+1}}I_{i+1}\,{}^{i+1}\omega_{i+1}
\end{aligned}$$

Inward iterations: $i : n \rightarrow 1$

$$\begin{aligned}
{}^{i}f_i &= {}^{i}_{i+1}R\,{}^{i+1}f_{i+1} + {}^{i}F_i, \\
{}^{i}n_i &= {}^{i}N_i + {}^{i}_{i+1}R\,{}^{i+1}n_{i+1} + {}^{i}P_{C_i} \times {}^{i}F_i + {}^{i}P_{i+1} \times {}^{i}_{i+1}R\,{}^{i+1}f_{i+1}, \\
\tau_i &= {}^{i}n_i^{T\,i}\hat{Z}_i
\end{aligned}$$

The effect of gravity loading on the links can be included by setting ${}^{0}\dot{v}_0 = G$, where G is the gravity vector.

The terms used in the above equations have meanings as given below:

${}^{A}_{B}R$	is the rotation matrix describing points in frame B relative to frame A
f_i	force exerted on link i by link $i-1$
n_i	torque exerted on link i by link $i-1$
τ_i	torque exerted at joint i
$\hat{Z}$	for finding $\hat{Z}$ component of a vector
F_i	Force acing at the center of mass of link i
N_i	Torque acting on link i
${}^{i}P_{i+1}$	Distance between the origin of the frame $i+1$ and frame i measured in the frame i
${}^{i}\omega_i$	angular velocity of link i with respect to frame i
${}^{i}v_i$	linear velocity of link i with respect to frame i
${}^{i}\dot{v}_{Ci}$	linear velocity of the centre of mass of the link i with respect to frame i.

In symbolic from, all the terms appearing in the equations are treated like variables. Given the values of some of the variables, values for the others can be calculated. This way it can be used in both forward dynamics and inverse dynamics situations. The main feature of symbolic formulation is reduction in simulation time. This is typically done by evaluating common subexpressions. The values of common subexpressions need to be calculated only once and can then be substituted in the equations. The final system of equations takes the form,

$$Ax = b \tag{3.9}$$

where A is a $2n \times 2n$ matrix, giving the mass description of the body in terms of masses of links and inertia. Each element of the matrix is represented in symbolic form, and can be calculated at each time step. b is a vector having $2n$ components. Each component of the vector represents force acting at the centre of mass of some link, or torque acting on the link.

When one link moves, due to application of some external force it exerts a force on the neighbouring links due to the joint constraint attachment. This force has to be added into the external force acting on the neighbouring links. Other causes of external force would be collision with the ground and of course the gravitational force. Gravitational force is constant and depends on the mass of the link. We shall discuss collision force computation in detail in the next chapter.

The torque acting on a link causes rotational motion. As we are considering only rotary joints, the only manner by which we can control the motion of the body is by varying (controlling) torques at the joints. Application of joint torque directly results in the rotary motion of the links connected at that joint, which is further passed on to the other links as well. This joint torque can be directly added to the other torques acting on the link. The other cause of torque on a link is the force acting on the links. The torque caused by a force depends on where the force is applied. It is given by the relation

$$\tau = p \times f$$

where p is the perpendicular distance of the joint at which torque τ is calculated, from the point of application of force f.

In the above equation 3.9, x is a $2n$ vector of linear and angular accelerations. Assuming that the forces and torques to create the motion are known, equations 3.7 and 3.8 are solved for linear and angular accelerations using the LU decomposition technique.

3.3.3 Integrating the equations

At a very high level, one can view the simulation as the process of numerically solving the ordinary differential equation (ODE)

$$\frac{dY(t)}{dt} = f(Y(t), t) \tag{3.10}$$

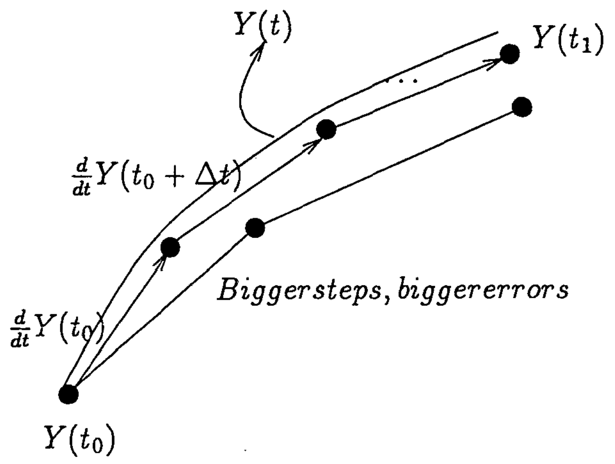


Figure 3.2: A numerical integration process

which describes the evolution of the system over time. The vector $\mathrm{Y}(t)$ describes the state of the system at time t given by

$$\mathrm{Y}(t) = \begin{pmatrix} x(t) \\ \theta(t) \\ v(t) \\ \omega(t) \end{pmatrix} \tag{3.11}$$

where
$x(t)$– represents the position of the centre of mass
$\theta(t)$– represents the orientation
$\mathrm{v}(t)$– represents the linear velocity
$\omega(t)$– represents the angular velocity.

Given the state of the rigid body at time t_0, numerical integration is used to advance the state from $\mathrm{Y}(t_0)$ to a new state $\mathrm{Y}(t_0+\Delta t)$ (cf Figure 3.2). Numerical integration technique requires evaluation of the right hand side of equation 3.10 for a particular value of t. This, in turn, requires computing the total forces $\mathrm{F}(t)$ and torque $\mathrm{N}(t)$ acting on the object at that instance.

Euler Method

The Euler method is a numerical integration technique which is one of the simplest to understand and easiest to implement. The Euler method assumes that the accelerations will be constant during the time step Δt of the integration. The integration equations are as follows.

For linear motion:

$$v(t + \Delta t) = v(t) + \dot{v}\ (t)\Delta t$$
$$x(t + \Delta t) = x(t) + v(t)\Delta t$$

For rotational motion:

$$\omega(t + \Delta t) = \omega(t) + \dot{\omega}\ (t)\Delta t$$
$$\theta(t + \Delta t) = \theta(t) + \omega(t)\Delta t$$

Stability Issues

Any numerical method for solving ordinary differential equations works on well-behaved linear differential equations. The problem arises when the differential equations are nonlinear or stiff or have discontinuities. All these conditions are likely to be present in our case. Our differential equations are nonlinear (second order). There are discontinuities due to collision with the ground. The stiffness problem also exists as discussed below. In these cases the numerical methods may fail to find the correct solution. This failure is indicated by instability in the method. An instability occurs when the numerical method behaves in an inconsistent manner. This inconsistency may show up as radically different solutions for different step sizes. Instabilities can be caused either by the differential equation, or the numerical method used for their solution or a combination of both. The most common situation is the combination of stiff differential equations and a numerical method (like Euler's) which cannot handle stiffness.

Stiffness

Stiffness in ordinary differential equations is the first known cause of instability in numerical methods. A set of differential equations is called stiff when in the solution, some components are slowly varying, while others have a very high frequency but quickly decay to zero. Since sampling of the differential equation is done at different time steps, these two components could become mixed, even after the high frequency parts have been reduced to zero. This causes the rapidly varying perturbation in a solution that has become unstable.

In case of dynamics the derivatives taken with respect to position and velocity (both linear and angular) affect the stiffness of the equations. Any force or torque that is not a function of position or velocity does not influence the stiffness of the equations. For example, stiffness will be introduced due to friction since frictional force is a function of velocity of the body. Similarly, a force or torque that is used to produce a required body orientation or enforce a constraint influences the stiffness. Since these are the techniques which are used here in controlling the motion of the body, they are bound to adversely influence the stiffness of the equations. Without the control forces and torques, the equations of motion for articulated bodies are not stiff, it is just when we control them that the stiffness gets introduced.

One possible solution to the stiffness problem is to use a small step size, in the Euler method. With a small step size, the differential equations will be sampled often enough to handle high frequency components. There are two problems with this approach. First, the step size directly determines the time required for simulation. The smaller the step size, the more the simulation time. Second, when the step size is decreased, rounding errors in the computations increase. Eventually a limit will be reached where the rounding errors dominate the solution. If a differential equation requires a step size smaller than this limit, then the Euler method cannot be used for solving the differential equations.

Second Order Runga-Kutta Technique

The advantage of the Euler technique is that it makes only one evaluation of the function derivative. In the context of the simulator, evaluating the state derivative corresponds to solving the system of motion equations. So it is desirable to have as few derivative evaluations as possible. However the very reason for the Euler methods' inaccuracy is the fact that it approximates the function derivative over the interval $[t_0, t_0 + \Delta t]$ by the derivative at t_0. Obviously more derivative evaluations are required in the interval in order to obtain a more accurate approximation for the function derivative over the entire interval. The second order Runga-Kutta method uses two derivative evaluations in the time interval. It evaluates the derivative at t_0, the beginning of the interval and at $(t_0 + \frac{\Delta t}{2})$, ie. halfway down the interval. In terms of equations it can be expressed as:

$$\begin{aligned} k_1 &= \Delta t\ f(Y(t_0), t_0) \\ k_2 &= \Delta t\ f(Y(t_0) + \frac{k_1}{2}, t_0 + \frac{\Delta t}{2}) \\ Y(t_0 + \Delta t) &\doteq Y(t_0) + k_2 + O(h^3) \end{aligned}$$

This method gives second order accuracy.

Fourth Order Runga-Kutta Technique

The fourth order Runga-Kutta technique makes four derivative evaluations in the time interval $[t_0, t_0 + \Delta t]$, one at the beginning (t_0), two at trial midpoints $(t_0 + \frac{\Delta t}{2})$ and one at a trial endpoint. The final function values are calculated using these derivatives.

In terms of equations,

$$\begin{aligned} k_1 &= \Delta t\ f(Y(t_0), t_0) \\ k_2 &= \Delta t\ f(Y(t_0 + \frac{k_1}{2}), t_0 + \frac{\Delta t}{2}) \\ k_3 &= \Delta t\ f(Y(t_0 + \frac{k_2}{2}), t_0 + \frac{\Delta t}{2}) \end{aligned}$$

$$k_4 = \Delta t\, f(Y(t_0 + k_3), t_0 + \Delta t)$$

and,

$$Y(t_0 + \Delta t) = Y(t_0) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(h^5)$$

This method gives fourth order accuracy.

Adaptive Step Size Control

If the step size is fixed, a step size will have to be chosen which is small enough to accommodate a rapidly changing function. This would be wasteful if the function is not rapidly varying. Any good ordinary differential equation (ODE) integrator must therefore exert some adaptive control over its own progress, making frequent changes to its step size. The purpose of this control is to achieve some predetermined accuracy without undue computational effort. In regions of the function domain where the function changes rapidly, small step sizes should be used while in regions where the function changes slowly, larger step sizes could be used.

3.4 Constraints

There are essentially two broad categories of constraints which the synthesized motion has to satisfy. These are:

1. Constraints due to the requirement of physical correctness
2. Constraints for enabling the animator to have control over the synthesized motion, such that it appears realistic and natural.

Most methods will try to restrict the search space by setting up constraints such as joint angles or other dependencies in state variable, reducing the size of the state space. Constraints are essentially in the form of mathematical equalities or inequalities. Any one or more of the variables appearing in the optimization problem could be constrained with the help of constraint equations. For example a constraint on the object trajectory will have the form:

$$C(x(t)) = 0$$

or

$$C(x(t)) \geq 0$$

The inequality constraints are the most general as all equality constraints can be treated as special cases of inequality constraints. For example, $C(x(t)) = 0$ can be replaced by $C(x(t)) \geq 0$ and $-C(x(t)) \geq 0$

3.4.1 Physical Correctness Constraints

These are the constraints which are mandatory from the point of physics. In general, unless these constraints are satisfied, no physical realism is possible. Most of the time these constraints are derived from the the physical description of the creature and its environment. We shall describe now three such constraints.

The Dynamic Equations of Motion

Given the mass of an object and forces and torques acting on the object, the motion of the object can be constrained by the equations of motion. Any motion which is not as per these equations of motion cannot be considered as physically correct. In all physically based animation techniques satisfying the dynamic equations of constraint is a primary goal.

Non Penetration Constraints

This is another constraint required for physically correct motion and arises from the fact that two rigid bodies cannot interpenetrate each other. Techniques for the handling of constraints of this kind are discussed in detail in the next chapter.

Structural Constraints

These are again constraints that have to be satisfied for physically correct motion. A given geometrical configuration like an articulated body cannot have its contact between two links broken any time during the motion. Joint connectivity constraints can be imposed in the form of equality constraints to be satisfied by the positions of the connected end points of the individual links. More often these constraints are automatically resolved by the manner in which the equations of motion are formulated.

3.4.2 Animator Specified Constraints

These constraints are essentially specified by the animator to control the motion that is synthesized. Most of these constraints are based on observations of similar movement in real life, say for example a limit on the maximum speed of movement. While there is no fixed set of constraint types, we describe below many of the constraints that have been applied in the different approaches.

Value Limiting Constraints

This is a simple and common type of constraint. Basically the animator limits the range of values for a parameter in the search space.

$$x_l \leq x(t) \leq x_m$$

In state space this would amount to putting limits on the position, velocity or acceleration of an object. In joint space this puts a limit on the joint angles and the angular velocities and angular acceleration and in force/torque space on forces and torques acting at the joint. For motion controllers these constraints would depend upon the parameterization that is applicable for the motion controller. In the pose control graph based controllers, for example this would impose limits on spring constants, rest lengths and transition times. For the banked stimulus response controllers, this would also put limits on the sensor variable values. It is important to recognize that most of these limits would be such that they are reasonable for real life movements of the same kind and are essential if the synthesized motion has to appear realistic. Applications of unnaturally large forces or impossible speeds would make the resulting motion look unrealistic.

Dynamic Constraints

The idea of dynamic constraints is to allow the animator to interactively specify geometric constraints on the motion such that the objects obey Newton's laws and at the same time obey the user specified geometric constraints. For example, an animator might constrain an object to move along a specified path or require two objects to remain at a specified distance apart. However, what constraint to specify in order to get the desired motion requires intuition of the mechanical aspect of motion. As such, the method is not suitable for motion synthesis of an active articulated body.

Space Time constraints

Spacetime constraint method is a useful technique for creating goal directed motion of an articulated body. A typical example of spacetime constraint is that the arm must be in particular position at particular time t_0, requiring it to be at some other position at time t_1. This specification alone is not sufficient to obtain the intermediate trajectory. In addition to spacetime constraints, an animator must specify an objective function, such as to perform the task with minimum energy or some other performance criteria.

Smoothness Constraints

The most natural motion results when the least amount of effort is put into controlling the motion. Therefore, when a large motion control problem is broken into a sequence of piece-wise control problems, it is necessary to impose continuity constraints such that the position and velocities are smooth at the boundary of each subinterval. One method of doing this is to impose additional constraints that the control function be a higher order continuous function of time through out the interval and then minimize the energy in control as well as its first derivative [15].

3.4.3 Constraint Handling Techniques

One common technique for solving the constrained optimization problem involves replacing the performance metric J with a *Lagrangian function* [57],

$$L(x, \lambda) = J(u) + \sum_i \lambda_i c_i(x, t)$$

The λ_i associated with each constraint c_i is called a *Lagrangian multiplier*. The Lagrangian multipliers roughly weight the influence of the constraint on the optimal solution value with inactive constraints having zero-valued multipliers. The *Kuhn-Tucker* [67] conditions provide optimality conditions for these and are given by:

$$\nabla_x L(x^*, \boldsymbol{\lambda}^*) = 0$$

$$c(x^*) \geq 0$$

$$\boldsymbol{\lambda}^* \geq 0$$

$$\boldsymbol{\lambda}^{*T} c^* = 0$$

where $\boldsymbol{\lambda}$ represents the vector of Lagrange multipliers and c represents the constraint equations. Variables superscripted by $*$ represent their values at optimality. These conditions state that all components of the gradient of the Lagrangian are zero, which will also implicitly satisfy the constraint. The system does not have a closed form solution. The numerical methods used in solution are typically of iterative nature, where an initial guess for a solution is improved upon at each iteration until an optimality criterion is reached. It is important to note that there is no guarantee that the solutions found are true minima or maxima. In fact, most of the time solutions found are likely to be approximate local optima due to the nonlinearity and vast size of search space. All methods based on the use of local optimization techniques essentially handle constraints using the above strategy. There is a very basic problem with this strategy. Since the constraints are integrated into the performance metric to be optimized and since the methods only try to optimize the metric value, the method as such cannot guarantee that the final solution obtained fully satisfies all the constraints. Certainly the performance metric is so formulated that constraint satisfaction is encouraged but not necessarily guaranteed. This is particularly disturbing when one wants physically correct solutions satisfying the constraints put by the dynamic equations of motion.

However, in the global optimization techniques using evolutionary methods, this problem does not exist. In all these approaches the population of solutions is chosen in such a fashion that all the constraints are satisfied *a priori*. Further, since the optimal motion is found in a space like the actuator space, and the the final motion is obtained by solving and simulating the dynamics equations of motion, physically correct motion is guaranteed.

3.5 Optimization Techniques

Classical numerical optimization typically use Sequential Quadratic Programming (SQP) [32] or projected gradient methods [88] to solve the optimization problem. Many of these are applicable to well behaved continuous functions which rely on using information about the gradient of the function to guide the direction of the search. If the derivatives of function cannot be computed, say, because it is discontinuous, these methods often fail. Such methods are generally referred to as *hillclimbing*. They can perform well on *unimodal* functions. But on *multimodal* functions they suffer from the problem that the first peak found will be climbed, and this may not be the highest peak. Having reached the top of a local maximum, no further progress can be made. A one-dimensional example is shown in Figure 3.3. The hillclimbing starts from an initially guessed point, say, X and moves are made to climb the hill until the peak at B is reached. However, the higher peak at C may not be reached ever. The method suggested by Witkin and Kass [108] minimizes energy. Their method, uses linear approximations for the constraints and quadratic approx-

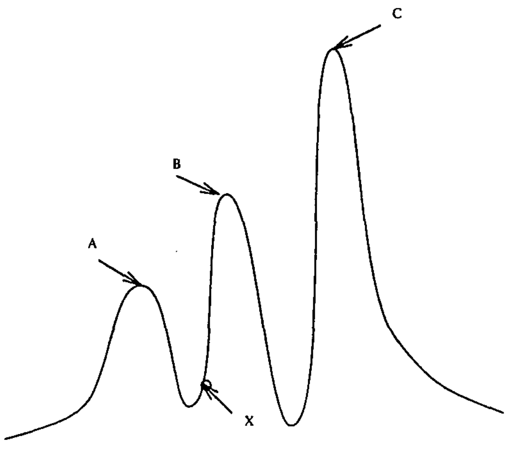


Figure 3.3: The hillclimbing approach

imation for the Lagrangian [32]. The method works by progressively refining an initial motion trajectory specified by the animator (cf Figure 3.4). On the

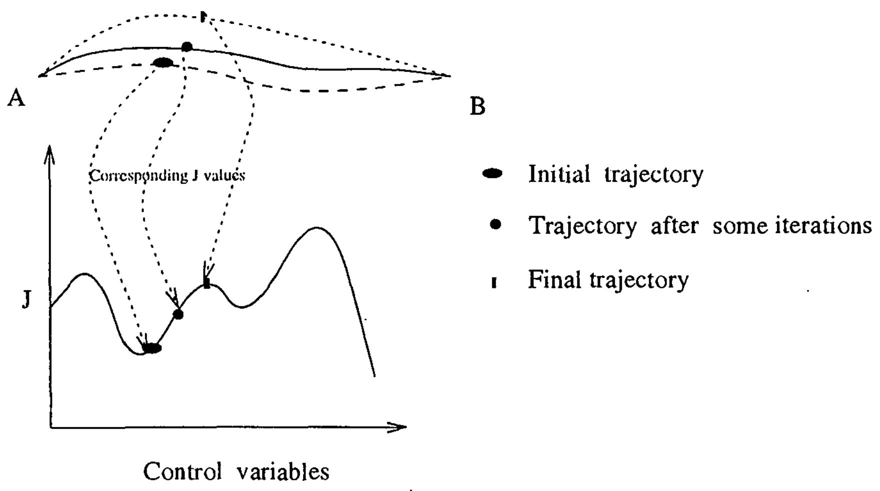


Figure 3.4: Space-time constraints method

other hand Brotman and Netravali [15] approximate the dynamics to a linear system given by the equation:

$$\dot{\mathbf{x}}\ (t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t)$$

where $\mathbf{A}(t)$, $\mathbf{B}(t)$ are $n \times n$ and $n \times m$ matrices with time varying elements. The overall performance metric J is defined with $J = \sum_0^{N-1} J_i$, where J_i is evaluated at each of the N subintervals between adjacent key poses at times t_i

and t_{i+1}. This reduces a multi-point boundary value problem to a series of two point constrained optimization problem. The individual performance metrics J_i are chosen to produce smooth trajectories and minimal control energy and is defined by

$$J_i = \int_0^T \left[\alpha \left|\dot{x}\ (t)\right|^2 + \beta \left|u(t)\right|^2\right] dt$$

α and β are weights for adjusting the balance of various motion characteristics in the final motion.

Like in key-frame interpolation a major advantage of this method is its ability to specify motion requiring precisely timed actions. The major difference between these methods and key-frame animation is, in the method of interpolation. In these methods geometric interpolation is replaced by a process that attempts computing physically correct trajectories automatically. As a result fewer key-frames are required. Further, the trajectories are goal oriented, that is, chosen to be optimal according some user-supplied fitness function.

These local optimization techniques were useful to introduce optimization as a useful mathematical tool for finding near physically correct motions with lesser requirements put on animator skills. However, they suffer from several computational problems. We discuss some of these problems below.

1. The systems to solve have many constraints and state variables, making the process of motion synthesis slow and non-interactive.

2. Since the methods use local optimization techniques, where an initial guess for a trajectory (in the form of key-frames) is improved upon at each iteration until an optimality criterion is reached, there is no guarantee that the solution obtained would be the best or as desired by the animator (cf Figure 3.4).

3. Most of the techniques are useful only when the animator can supply a good guess of the motion trajectory. At times specifying such a trajectory is as burdensome as specifying key-frames.

4. Many a times the search space is large, multimodal and discontinuous

(due to collisions) as a result these gradient based search techniques fail to obtain a good solution.

It is therefore clear that we need a mechanism which can start the search effectively through a large multimodal search space. It should avoid achieving the false peaks and should not require the function to be continuous in order to proceed. The two most common methods which are suitable for this kind of situations are *simulated annealing* and *genetic algorithms*. Both are stochastic optimization techniques, which are based on natural processes found in nature. Simulated annealing is patterned after the physical process of annealing found in metals, where as genetic algorithms are based on the principle of evolution found in nature.

3.5.1 Simulated annealing

Simulated annealing [58] is a computational process where the exploration of the whole space is done early on so that the final solution is relatively insensitive to the starting state. In the physical process of annealing, the physical substances such as metals are melted (*i.e* raised to high energy levels) and then gradually cooled until some solid state is reached. The goal of this process is to produce a minimal energy final state. The computational equivalent of energy is the objective function. In an annealing process, as the temperature decreases, the probability of a large uphill move is lower than the probability of a small one. Thus such moves are more likely to occur during the beginning of the process when the temperature is high, and they become less likely at the end when the temperature becomes lower, whereas downward moves are allowed any time. The rate at which the system is cooled is called as the annealing schedule. If the annealing schedule is too rapid, the method is subject to stagnation in a local minima (it degenerates into a pure descent method). If the annealing schedule is very slow the method becomes impractical for generating suitable solutions to complex problems within reasonable computational limits. In physically based animation, Van de Panne and Fiume have used simulated annealing technique for global optimization. Using the technique they have synthesized the weights of sensor actuator networks, which influence the movement control of virtual creatures.

3.5.2 Genetic Algorithms

Genetic Algorithms (GA's) are adaptive methods which may be used to solve general search and optimization problems. The basic principles of GAs were laid down rigorously by Holland [51] and are well described in many texts, *e.g* Davis [24], Goldberg [36], Michalewicz [74]. GAs use a direct analogy of evolution according to the principles of natural selection and survival of the fittest. They work with a population of individuals each representing a possible solution to a given problem. Each individual is assigned a "fitness score" according to how good a solution to the problem it is. The highly fit individuals are given opportunities to "reproduce" by "cross breeding" with other individuals in the population. This produces new individuals as "offsprings" which share some characteristics with each "parent". The least fit members of the population are less likely to get selected for reproduction and so "die" out. In this way, over many generations good characteristics are spread through out the population as they evolve. By favouring the mating of the more fit individuals, the most promising regions of the search space are explored. If the GAs have been designed well, the population will *converge* to an optimal solution to the problem. The population is said to have converged when 95% of the individuals in a population share the same value of the solution.

Algorithmically the process can be described as follows. Let $P(t) = x_1^t, \ldots, x_n^t$ be a population (set) of solutions, for iteration t. Each solution x_i^t is evaluated to give some measure of *fitness*. Then a new population is formed for iteration $t + 1$ by selecting more fit individuals. Some members of the new population are involved in reproduction by means of *genetic* operators to form new solutions. There are unary transformations m_i (mutation types), which create a new individual (solution) by a small change in a single individual, and higher order transformation c_j (crossover type), which create two new individuals by combining parts from two other individuals. After some number of generations the algorithm converges. The best solution is assumed to and often represents the optimal solution.

```
procedure genetic algorithm;
begin
    t :=0;
    initialize P(t);
    evaluate P(t);
    while not ( terminate condition) do
    begin
        t := t + 1;
        select P(t) from P(t-1);
        recombine P(t);
        mutate P(t);
        evaluate P(t);
    end;
end;
```

Genetic algorithms work well even if the search space has some of the difficult to handle properties mentioned above. Considering a population of points rather than a simple point at start, GAs climb many peaks parallelly. Thus the probability of finding false peak points is reduced in the case of *multimodal* (many-peaked) search spaces. In order to proceed, GAs only require objective function values associated with individual solutions, so there is no need for the search space to be continuous or differentiable as no derivative needs to be used to climb the peak [1].

Classical genetic algorithms use a bit string for representation of a solution. The whole probabilistic theory of GAs is based on this representation. Genetic

[1]as required in gradient based search techniques

operators are well defined for binary representation and are unique for all types of problems. A problem with the bit representation is that the structures are difficult to impose on the parameters.

In motion optimization, we have floating point numbers to deal with. In order to use GAs they need to be converted to binary representation with precision information. Also our solution representation requires structures on the parameters. Hence genetic algorithms are not very practical in their canonical form for the kind of optimization problems encountered in physically based animation. A variant of genetic algorithms which use floating point representation are called *Evolutionary programming algorithms.*.

The genetic algorithms differ from evolutionary programming in the following aspects [28]:

1. In genetic algorithms a binary coding of the parameters to be evolved is used, not the floating point parameters themselves.
2. The number of offsprings to be created from each parent is proportional to the parent's fitness relative to all other members of the current population.
3. Parents create offsprings through the use of specific genetic operators such as one-point crossover, and bit mutation.

On the other hand in evolutionary programming [28]:

1. Rather than a single coding structure to every problem, each problem is regarded as unique.
2. Successful simulations need not create more than a single offspring per parent.
3. Offsprings are created through many mutation operations that follow naturally from the chosen problem representation. No emphasis is placed on the use of a crossover operation.

Many of the original proposals in GA have undergone significant revision since first definition. Much of the current research in GA has foregone the use of bit strings [93]. Experiments have indicated the superiority of hybrid crossover mechanisms to the one-point crossover originally proposed by Holland. Some of the research has indicated a greater role for mutation in evolutionary search and has illustrated cases where crossover can be detrimental to search. The major advantage of evolutionary programming is that, we can impose a structure on the parameters. However, it requires one to define *mutation* and *crossover* operations. These operators are very problem specific. Although there is no proven theory behind the working of this approach, it works well if operators are well defined.

Many evolutionary programing algorithms can be formulated for a given problem. Such programs may differ in many ways. They can use different data structures for representing the individual solutions, different genetic operators for transforming individuals, methods for creating an initial population, methods for handling constraints of the problem, and parameters (population size, probabilities of applying different operators etc.). However, they share a common principle; a population of individuals undergo some transformations, and during this process of evolution the individuals strive for survival and only the fittest survive. In physically based animation, Ngo and Marks [81] have used a massively parallel genetic algorithm to synthesize motion of articulated figures with banked stimulus response controller. A genetic algorithm is also used for controller synthesis in the recent work by Sim [89] and by Gritz and Hahn [37].

3.6 Remarks

The initial techniques which used local optimization procedures were primarily responsible for introducing optimization as the tool for motion synthesis in computer animation. There are a number of problems inherent in the local optimization methods. Specifically, the amalgamation of constraints including the dynamic equations of motion into the objective function to be optimized has the undesirable side effect that the synthesized optimal motion does not necessarily satisfy the constraints and hence need not even be physically realizable. It must be said here that local optimality would surely mean that constraint satisfaction and physical correctness would be encouraged but not guaranteed. Use of global optimization methods is therefore mandated. Present algorithms based on the genetic programming paradigm seem to be the most suited candidates. While such algorithms do search for the optimum globally they too do not guarantee that the true optimum can always be found. But probability is certainly much greater. Such algorithms are simple to program but are computation intensive. One possible solûtion is to use parallel computation techniques. This approach has been adopted by us in our implementation and the parallel algorithm is described later in chapter 6.

Chapter 4

Modelling Object Interaction

Articulated figures representative of living creatures locomote themselves not only through internally generated forces and torques but also with the help of *external forces* that arise through interaction with other virtual objects as the articulated figures move in the virtual environment. If no special attention is paid to object interaction, these objects will move through each other producing unrealistic and disconcerting visual effects. A primary requirement of physically correct animation is to model the dynamic effects of these interactions in the form of suitable forces and torques. Along with the internal forces and torques [1] these are then incorporated into the dynamic equations of motion for obtaining the desired movement.

Methods for generation of internal forces and torques have been discussed in detail in the earlier chapter. In this chapter we discuss methods for the modelling and synthesis of external forces. There are essentially two kinds of external forces that a rigid object can experience:

[1]Some times the term internal forces has been used to distinguish constraint forces like the one that has to be generated for maintaining joint connectivity from external forces like gravity. In this thesis however, by external forces and torques we mean all environmental and environment induced forces/torques, and we reserve the term internal forces/torques for those whose generation is internally motivated in the creature in order to achieve a goal oriented movement.

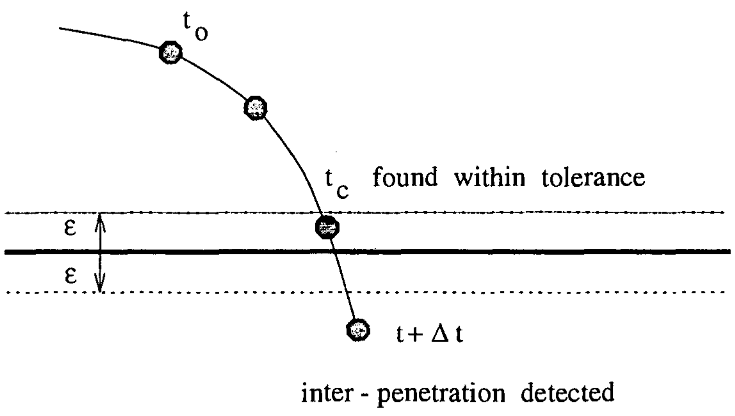


Figure 4.1: The particle colliding with the floor

- Field forces like gravitational force, that are always present throughout the virtual environment. A gravity like force is very simple to model and can be easily incorporated into the equations of motion.
- Constraint forces that arise out of bilateral constraints and unilateral constraints such as joint connectivity, collision and contact constraints.

Bilateral constraint forces typically arise in representing idealized joints which connect one rigid body to another. Bilateral constraints are explicitly specified as part of the description of the articulated body. An example of a bilateral constraint is a rotary joint. Typically, bilateral constraints are valid through out the simulation unless they are relaxed explicitly during the course of simulation.

Collision and contact constraints on the other hand come and go as objects move about and interact with other objects in the environment. For example, consider a point mass being dropped on to the floor under the influence of gravity (cf Figure 4.1). Assuming rigid body behaviour for both particle and floor, it is clear that we cannot allow the particle to penetrate below the floor when it strikes the floor. This means that at the very instance that the particle actually comes into contact with the floor, an abrupt change in its velocity has to occur, making it move away from the floor.

However a different approach has to be taken for a flexible object. For a flexible

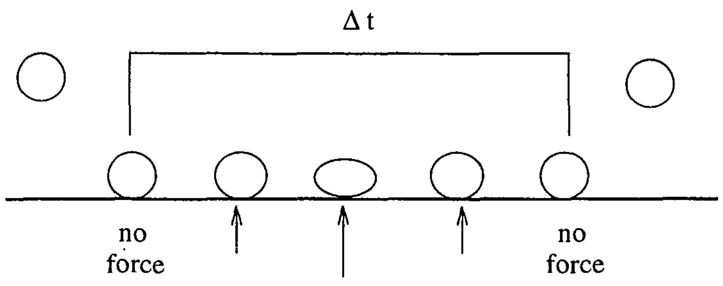


Figure 4.2: A rubber ball colliding with the floor

object such as a rubber ball, the collision can be considered as occurring gradually. Over some fairly small, but finite span of time, a force would act between the ball and floor and change its velocity (cf Figure 4.2). During this time the ball would deform, due to the force. The more rigid the ball is, less would it deform and shorter would be the time of contact. In the limiting case, the ball is infinitely rigid, does not deform, and unless the ball's downward velocity is instantaneously reversed, it will penetrate the floor (cf Figure 4.3). Thus for rigid body motion under constraints, we have two basic types of contacts to deal with. When two bodies at the point of contact, have a relative velocity towards each other, it is called a colliding contact. Whenever, the relative velocity between the bodies at the point of contact is zero, the bodies are said to be in resting contact. Like, the rubber ball bouncing off the floor, in resting contact, objects remain in touch with each other over a finite span of time. Although, there is no deformation due to bodies being rigid. Detecting whether objects are in colliding contact or resting contact can be considered as purely a kinematics problem depending only on position and velocity. However, after the collision, how the objects should move is decided by the dynamics. It must be noted here that both colliding and resting contacts may also require the modelling of frictional effects.

4.1 Motion Simulation with constraints

As we have seen earlier in Chapter 3, at a very high level, one can view motion simulation as the process of numerically solving the ordinary differential

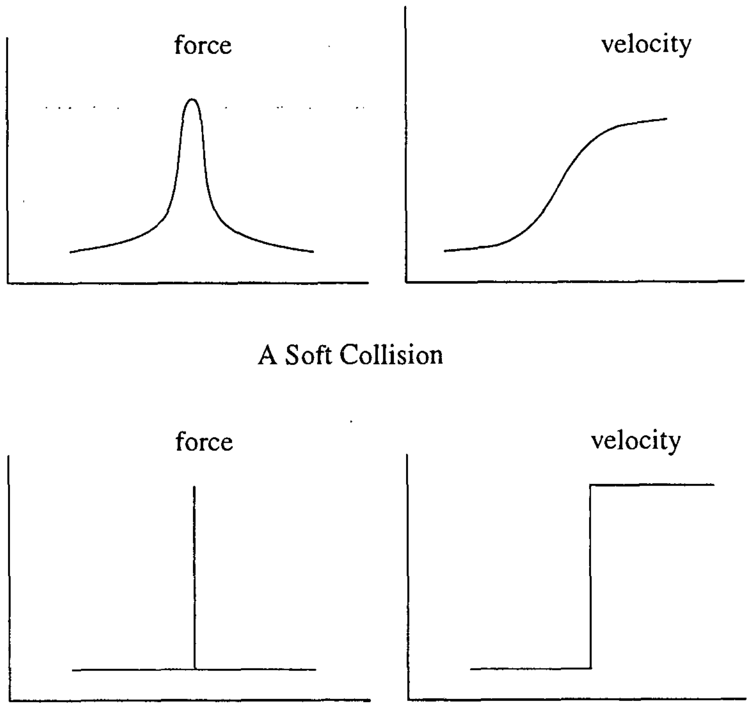


Figure 4.3: Soft body collision v/s Rigid body collision

equation (ODE)

$$\frac{d\mathrm{Y}(t)}{dt} = f(\mathrm{Y}(t), t) \tag{4.1}$$

Where the vector $\mathrm{Y}(t)$ describes the state of the system at time t. For a rigid body motion, the state $\mathrm{Y}(t)$ is defined as:

$$\mathrm{Y}(t) = \left[\ x(t)\ \theta(t)\ v(t)\ \omega(t)\ \right]^T \tag{4.2}$$

where	$x(t)$	– represents the position of the centre of mass,
	$\theta(t)$	– represents the orientation,
	$\mathrm{v}(t)$	–represents the linear velocity, and
	$\omega(t)$	– represents the angular velocity.

Given the state at time t_0 i.e $\mathrm{Y}(t_0)$ and $f(\mathrm{Y}(t_0), t_0)$, the simulation system uses numerical integration to advance its state from $\mathrm{Y}(t_0)$ to $\mathrm{Y}(t_0 + \Delta t)$. Numerical integration techniques require evaluating $f(\mathrm{Y}(t_0), t_0)$, which in turn requires computing the force $\mathrm{F}(t_0)$ and $\tau(t_0)$ acting on an object, representing both external and internal forces. When there are no constraints on the object's motion (that is, no obstacles to encounter, as in objects in flight), then all these forces and torques can be determined trivially and the simulation is a simple and straightforward numerical integration of Equation 4.1. In the presence of obstacles however the numerical integrator has the following problems:

- Firstly, we may have the situations that at an intermediate state $\mathrm{Y}(t_i)$ objects do not penetrate while in the state immediately following it, that is, $\mathrm{Y}(t_i+\Delta t)$, objects may penetrate. Unfortunately, the differential equation contains no information about the geometry of objects in the simulation. Therefore, we cannot determine when collisions will occur, solely on the basis of the differential equations.

- Secondly, a colliding contact requires an instantaneous change in velocity, say at the instant that the collision occurs. As a result the numerical integrator that solves the differential equation, suffers from discontinuity in state variable $\mathrm{Y}(t_c)$ at the time of collision (cf Figure 4.4).

The standard technique to overcome these problems consists of testing the collision outside the numerical integration loop and solving the differential equation

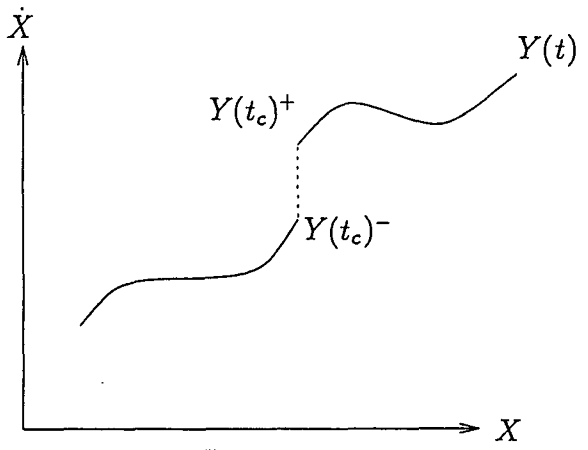


Figure 4.4: A discontinuity in the state variable due to collision

piece-wise. A piece-wise solution involves, stopping the solver at the instant the collision is detected, computing the new velocities, and reinitializing the state with new velocity variables. It is important to note that the state $Y(t_c)^-$ just before collision and the state $Y(t_c)^+$ just after the collision should agree for all spatial variables (position and orientation) but may differ in velocity variables. The various methods and issues in computing the exact time of collision t_c, is discussed next.

4.2 Collision and Contact Detection Techniques

There are two ways to look at the collision/contact determination problem. First, as a continuous function of time from time t_0 to t_i. Given this view point, the basic problem to be solved is "at what time" and "where" do bodies first come into contact? Second, the problem can be considered discretely, at a sequence of time values $t_0 < t_0 + \Delta t_1 < t_0 + \Delta t_2 \ldots$ In this viewpoint, the basic problem is given the position of bodies at time t_i and time $t_i + \Delta t$, and the fact that penetration has occurred during the interval Δt, where do bodies interpenetrate and contact each other.

At the very core, collision/contact detection is a spatial interference problem which has been extensively studied in the fields of computational geometry

[85], and robotics [17]. In computational geometry the problem is solved for a static environment. That is, given two objects one has to determine whether the objects intersect. The emphasis is on complex object shapes and exact intersection computation. In robotics on the other hand, the collision detection problem is solved as a dynamic environment problem. That is, given two objects and their paths, the problem is to determine whether the objects will collide and if so at what time do the bodies first come into contact. Examples include an algorithm for determining the first collision between rigid polyhedral objects [17]. Making the assumption of constant angular velocity, the problem is reduced to that of determining the first instant of collision to the problem of finding roots of polynomials. Since no closed form solution exists for the time at which the first intersection occurs an iterative numerical method is used to determine time.

In physically based simulations however, the paths of the colliding objects are not known in advance. In fact they are to be determined. These paths depend very much on the interaction of the moving object with other objects in the environment as well as the objects own internal forces. Von Herzen *et al.* [44] describe an algorithm that determines the first collision between parametrically defined time-dependent surfaces. Very recently, interval arithmetic based methods [77] have attracted considerable attention as a method to deal with collision detection. Duff [26] describes a collision detection method using interval analysis that handles rigid-body motion of implicit curved surfaces but with restrictions on the motion path. Similarly, Synder [92] uses interval analysis to find the first time of collision between both parametric and implicit time dependent curved surfaces. The methods based on continuum approach are computationally very expensive and are not really practical in todays general computing environments.

The second approach takes a discrete view to the collision detection problem. A straight forward method of collision detection would be to solve a sequence of static problems one per time step. For example, as we run the simulation, we compute the position and orientation of the object at times t_0, $t_0 + \Delta t$, $t_0 + \Delta 2t$

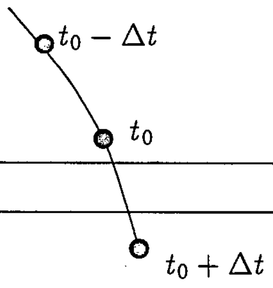


Figure 4.5: A missed collision

and so on [2]. Suppose all we know is that the time of collision t_c when objects first come into contact lies between t_0 and $t_0 + \Delta t$, then computing t_c involves a binary search in the interval t_0 and $t_0 + \Delta t$. That is, if at time $t_0 + \Delta t$ we detect inter-penetration, we inform the ODE solver that we wish to restart back at time t_0, and simulate forward to time $t_0 + \frac{\Delta t}{2}$. If the simulator reaches $t_0 + \frac{\Delta t}{2}$ with out encountering inter-penetration, we know the collision time t_c lies between $t_0 + \frac{\Delta t}{2}$ and $t_0 + \Delta t$. Otherwise, t_c is less than $t_0 + \frac{\Delta t}{2}$ and we try to simulate from $t_0 + \frac{\Delta t}{4}$ and so on. The method will terminate when the objects are found to collide within some tolerance. The binary search method is slow but is easy to implement and is very robust.

The problem with this simple method of collision detection is that these algorithms essentially ignore any geometric similarity that may exist between two consecutive states. Secondly, even if we find that both states at t_0 and $t_0 + \Delta t$ are legal, the method does not guarantee that a collision has not been missed (cf Figure 4.5). The recent work by Baraff [9] and Lin and Canny [63] has focused on collision detection algorithms for dynamic simulation that efficiently reuse previously computed information. In particular, Lin and Canny describe a collision detection algorithm for convex polyhedra that takes roughly $O(1)$ time to test a pair of polyhedra. Baraff describes a coherence based bounding box that detects overlap between n bounding boxes in roughly $O(n)$ time over the course of simulation. Methods for coherence-based collision detection among

[2]The ODE solver need not proceed with equal size time steps as explained in Chapter 3

convex curved surfaces are also described [9]. To guarantee that the object has had no collision within the time interval Δt, one must consider the entire path during the interval. But as was said earlier, in general this path is not known. The simplest way to approximate the trajectory is by linearizing the motion between time interval t_0 and $t_0 + \Delta t$. Another common assumption to solve this problem is to make sure that the velocities of the moving objects are small as compared to the time interval Δt.

Collision detection is in general computationally very expensive. A naive collision detection algorithm with n objects requires $O(n^2)$ comparisons at every time step of simulation. Bounding box based preprocessing has been proposed for increased efficiency. In these methods to improve the performance it is necessary to determine only pairs of objects which really require consideration by the collision detection algorithm. This can be done by enclosing each object in the simulation by a bounding box whose sides are parallel to the coordinate axis. Given objects A and B, if their bounding boxes do not overlap, there is no need to subject the objects to any further consideration. Moreover, this technique can be implemented hierarchically to reduce computation time [75]. Research on more efficient collision detection methods continues.

4.3 Collision Resolution Techniques

Once the collision has been detected and the exact points of contact between the colliding objects have been determined the next step is to ensure that the non-interpenetration constraint is maintained. Ideally, this is done by computing a constraint force at each contact point that acts in a direction normal to the contact surface at the point of contact and exactly prevents interpenetration. This would require the solution of a non-linear system of equations and is fairly complicated. The second method, called the penalty method is much less complicated, but does not completely eliminate interpenetration. Essentially a contact is modelled by placing a damped spring at the contact point. As the amount of interpenetration increases a repulsive force acts between the objects pushing them apart.

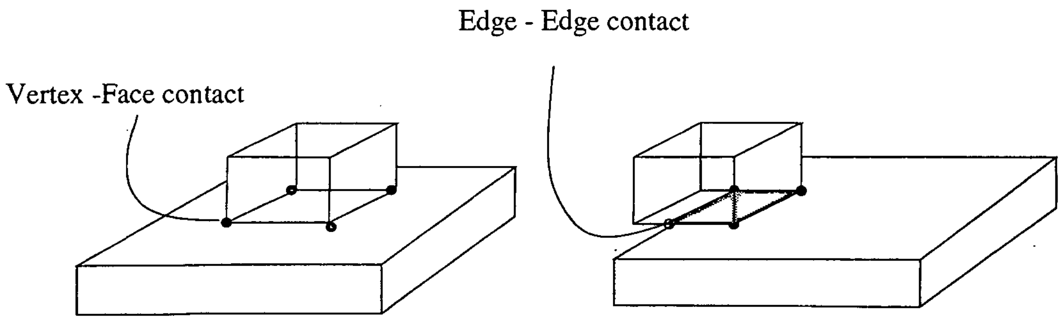


Figure 4.6: A contact between two polyhedra

4.4 Exact Methods

Once the exact time (within tolerance) of collision/contact is computed using a numerical or binary search, the next step is to find all the points of contact. For a collision between a point particle and a surface this is very simple, since nothing more than substituting the position of the particle in the equation of surface is required. However, for objects with complicated shapes, the problem is much more difficult. To simplify the matter, let us assume that all bodies are polyhedra, and every contact point between bodies has been detected. The contact point between bodies can be considered to be either vertex/face contacts or edge/edge contact. We shall assume that vertex/vertex and vertex/edge as degenerate cases and have to be handled separately in some *ad hoc* manner. A vertex/face contact occurs when a vertex on one polyhedron is coincident with a face of the other polyhedron (cf Figure 4.6). An edge/edge contact occurs when a pair of edges are coincident.

4.4.1 Colliding Contact

Consider two bodies A and B which come in contact at point P at time t_c. Let $P_a(t)$ and $P_b(t)$ denote the points on body A and on body B respectively (expressed in their own coordinate frame) such that $P_a(t) = P_b(t)$ at $t = t_c$. Although $P_a(t)$ and $P_b(t)$are coincident at time t_c, the velocities of the two points at contact time t_c may be quite different. If we denote $\dot{P}_a\ (t)$ and $\dot{P}_b\ (t)$ to be their velocities at time of contact, then the relative velocity in the direction

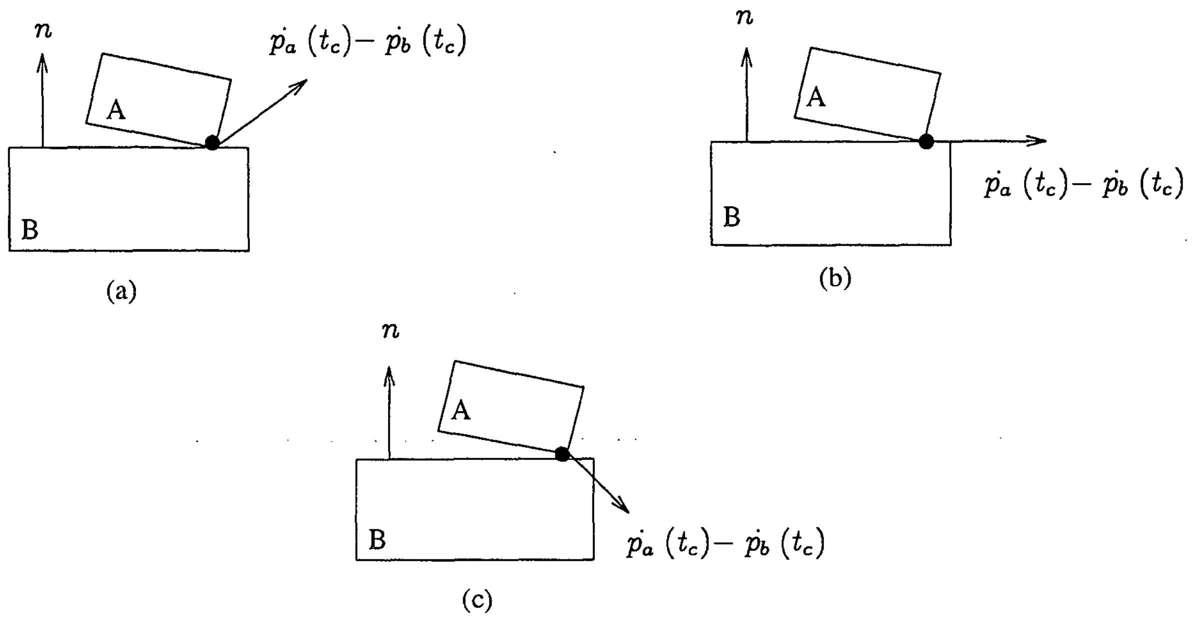


Figure 4.7: (a) Colliding contact (b) Resting contact (c) Separating contact

normal to one surface will be given by

$$v_{rel} = \boldsymbol{n}(t_c).(\dot{\boldsymbol{P}}_a\ (t_c) - \dot{\boldsymbol{P}}_b\ (t_c))$$

which is a scalar (cf Figure 4.7). In this equation $\boldsymbol{n}(t_c)$ is the unit surface normal defined for each contact point at t_c [3]. The quantity v_{rel} gives the component of the velocity in $\boldsymbol{n}(t_c)$ direction. Clearly if v_{rel} is positive, then it means that the bodies are moving apart, and that this contact point will disappear immediately after t_c (cf Figure 4.7). We do not now have to worry further about this case. If v_{rel} is zero, then, the bodies are neither approaching nor receding at the point of contact. This is exactly what we mean by resting contact. If on the other hand $v_{rel} < 0$, this means unless the velocities of the bodies undergo an immediate change, interpenetration will occur in the next time step (cf Figure 4.7). Since we want the bodies to change their velocities instantaneously we

[3]Assumption is that one can always find such a normal

apply an impulse force J in the direction of $n(t_c)$ [4].

$$J = jn(t_c)$$

An impulse is a vector quantity, just like a force, but it has units of momentum. Applying an impulse produces an instantaneous change in the velocity of a body. For example, if we apply an impulse J to a rigid body with mass M, then change in the linear velocity $\Delta v = \frac{J}{M}$. The magnitude j is computed by solving the empirical law for frictionless collisions such as:

$$v_{rel}^{+} = -\epsilon\, v_{rel}^{-}$$

where the quantity ϵ is called coefficient of restitution and must satisfy $0 \leq \epsilon \leq 1$. If $\epsilon = 1$, then $v_{rel}^{+} = -v_{rel}^{-}$, and collision is perfect and there is no loss in energy.

Moore and Wilhelms [75] and Hahn [39] have proposed a solution to the problem based on the conservation of linear and angular momentum. Fifteen linear equations are set up and solved for fifteen unknowns. These unknowns are the three components of the resultant linear, angular momentum and the impulse vector. Out of these fifteen equations, twelve equations are due to the momentum conservation principle:

$$m_1 v_1' = m_1 v_1 + R$$

$$m_2 v_2' = m_2 v_2 - R$$

$$I_1 w_1' = I_1 w_1 + \rho_1 \times R$$

$$I_2 w_2' = I_2 w_2 - \rho_2 \times R$$

where

- ρ_1, ρ_2 are vectors from the centre of mass of each object to the point of collision.

[4] This assumption is valid provided we are considering only frictionless systems.

as a function of unknown f_i's. This results into equations, which can be solved using a numerical technique called quadratic programming to determine the f_i's.

The constraint based approach computes exactly the non-penetration constraint forces that are required to cancel accelerations that would result in interpenetration. However the method typically requires solving nonlinear systems of equations, and is fairly difficult to implement.

4.5 Penalty Method

A vast number of simulations [61, 71, 75] have employed the penalty method to enforce non-penetration constraints. Applications include the simulation of deformable bodies, cloths, and articulated rigid bodies. The penalty method is a very attractive model in some respect, because it is extremely simple to implement and very versatile.

Unlike the exact methods discussed above, the penalty method provides an approximate solution to the collision/contact problem. It is based on a numerical solution method for constraint optimization where a constrained problem is converted to an unconstrained problem with deviation from the constraint being penalized. In the converted problem however, satisfaction of the constraint is encouraged, but not strictly enforced. This is illustrated below. A typical constrained optimization problem such as

$$\text{minimize } f(z) \text{ such that } g(z) = 0$$

can be rewritten as an unconstrained problem as follows:

$$\text{minimize } f(z) + kg(z)^2 \text{as } k \rightarrow \infty$$

The term $kg(z)^2$ is called the penalty function. The idea is that as k grows larger, potential solutions for z must make $g(z)^2$ smaller. In the limit, as $k \rightarrow \infty$, the

- m_1, m_2 are the object masses.
- I_1, I_2 are inertia matrices of the objects.
- v_1 , v_2 are the linear velocities of objects before the collision.
- v'_1 v'_2 are the linear velocities of objects after the collision.
- w_1, w_2 are the angular velocities of objects before the collision.
- w'_1, w'_2 are the angular velocities of objects after the collision.
- $\boldsymbol{R}$ is the impulse vector, by convention directed from object 2 to 1.

The last three equations depend on the collision behaviour, *i.e*, elastic or non-elastic collision, with or without friction, etc. The square linear system of fifteen equations in fifteen unknowns is solved by standard Gauss-Jordan or LU decomposition method.

4.4.2 Resting Contact

As was the case for colliding contact, in a resting contact too, at each contact point, there is assumed a contact force $f_i \boldsymbol{n}_i(t_c)$ that acts normal to the contact surface. Here f_i is an unknown scalar, and $\boldsymbol{n}_i(t_c)$ is the normal at the i^{th} contact point. The goal is to determine what each f_i is. In computing f_i's they must all be determined at the same time, since the force at the i^{th} contact point may influence one or both of the bodies. For resting contact, the f_i's are computed subject to three conditions. First, the contact forces must prevent inter-penetration that is, the contact forces must be strong enough to prevent two bodies in contact from being pushed towards one another. Second, the contact forces must be repulsive, that is, contact forces can push bodies apart, but can never hold bodies together. Last, the force at a contact point becomes zero if the bodies begin to separate. A simple function d_i is defined. d_i denotes the distance between two objects near the point of contact. In order to actually find f_i's which satisfy the three conditions stated above, we need to express d_i

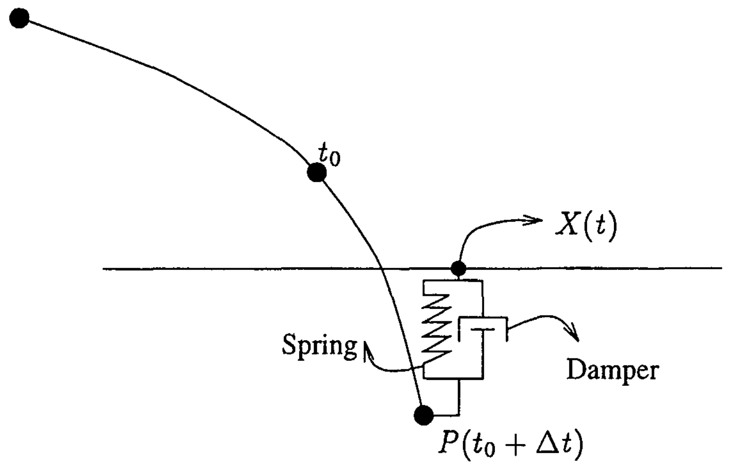


Figure 4.8: A collision between floor and a particle modelled using penalty method

solution to the problem satisfies $g(z) = 0$ while at the same time minimizing $f(z)$. In practice, z is obtained by solving the problem for a series of increasing values of k until the series of solutions converge (within numerical tolerance) to a limit. Although the method has a theoretically firm basis, in practice, it is not a very robust numerical method. The main problem is that as k grows, the problem becomes very poorly conditioned and difficult to solve. The main attraction of course is that it provides a very simple way of turning a constrained problem into an unconstrained one.

For a specific example, consider a collision between a particle and a surface (cf Figure 4.8). Using the method described above the problem can be converted to an unconstrained dynamics problem. As soon as collision is detected, a penalty force is applied on the particle, in a direction normal to the surface, so as to push the particle away from the surface. Typically, the penalty force is modelled as a linear spring force; that is, the penalty force pushes the particle away from the surface with a strength equal to some constant k times the distance of penetration. If we let $\boldsymbol{P}(t)$ denote the point on the object which has penetrated into the surface at time t, then the penalty force is computed as

$$-k\,(\boldsymbol{X}(t) - \boldsymbol{P}(t)\,)$$

where $X(t)$ denotes the closest point on the surface to the point $P(t)$. From the expression above one can see that larger the penetration larger the force $-k\,(X(t) - P(t)\,)$.

If the penalty method for dynamics were to completely emulate the penalty method for constrained optimization, the simulation would be repeated with increasing values of k until the behaviour of the particle approached a limit. However, the penalty method, as used by dynamics, chooses a single value for k. Setting the penalty constant too high significantly increases the cost of the simulation, while setting the stiffness too low can lead to an unacceptable degree of interpretation. Moreover, a good choice for penalty stiffness can vary greatly over the course of simulation, and it is usually impossible to make a reasonable prediction for a single suitable stiffness value. If k is small, it may not do an adequate job of enforcing the constraint; that is, the particle will penetrate the surface before being pulled away. Visually this may be very disconcerting and may be totally unacceptable to the animator. If k is large, it may give rise to a funny bounce which again may look unnatural. Further, as in other optimization problems, ill-conditioning occurs as k grows large, in the guise of "stiffness" of the differential equations of motion. As already discussed earlier, stiff differential equations are expensive and difficult to solve.

One of the biggest plus points of the penalty method is that it models collision response as a continuous time varying phenomena clearly requiring state $\mathrm{Y}(t)$ to change continuously when collision occurs. As a result there is no need to stop and start the differential equation solver with fresh initial values like that in the case of the analytical method. Further, it handles both colliding and resting contacts uniformly.

4.6 System with Friction

So far we had considered a frictionless system in which constraint forces and impact forces act normal to the contacting surface. This assumption is no longer valid for a system with friction. In fact, friction adds considerable complication to a rigid body simulation. For example, in a frictionless system the constraint forces that act to prevent interpenetration are conservative; that is, they perform no net work. Where as if we wish to model the effects of friction, we will need to compute friction forces that act tangentially to the contact surface to prevent or oppose sliding between objects at contact points and do work in the form of dissipation of energy. The classical friction model for contacting surfaces is the one given by Coulomb. This model suggests an empirical relationship between the normal force magnitude f_N and frictional force magnitude $f_x^2 + f_y^2$ in the form of

$$f_x^2 + f_y^2 \leq (\mu f_N)^2$$

where f_x and f_y are the components of frictional force in the plane normal to the contacting surface and μ is a coefficient of friction that depends on material properties, and may be different at each contact point. The relationship of these forces at each contact point also depends on whether or not bodies are currently sliding relative to one another or are at relative rest. If the tangential velocity is nonzero, then the friction force is called *dynamic*; otherwise, the friction force is called *static*. Typically, the coefficient μ of static friction μ_{static} is larger than the coefficient μ of dynamic friction $\mu_{dynamic}$.

From the Coulomb's empirical relation it is clear that there does not exist a unique relationship between constraint force and frictional force. Rather, it imposes inequality. This means that the law does not suggest an effective means of determining the contact forces, and in practice, simulations must occasionally search for a set of contact forces satisfying constraints. Given this state of affairs, it is not too surprising to find that the search might turn up more than one solution or fail to turn up any solution.

4.6.1 Collisions with Friction

When two bodies collide at a contact point with friction, the collision is modelled to take place over some small but nonzero time interval. During the time interval of the collision, the normal and friction forces must satisfy the condition laid by the Coloumb's empirical law. Frictional force at a single point of contact is computed by examining the limiting behaviour of the system as the time interval of contact is reduced to zero. Modelling simultaneous collisions is more complicated. Consider for example a cube dropped onto a level plane surface so that all four vertices of the bottom face of the cube strike the plane surface together. Lötstedt [65] computes simultaneous frictional impulses in three dimensions by using a modification of Coulomb friction law that causes impacts to dissipate as much as possible. However, in general, it is unclear as to how to deal with simultaneous impacts with friction and the problem is still open.

4.6.2 Contact with Friction

Finding contact forces that satisfy the Coulomb friction model at the contacting point is also extremely difficult. Unlike the frictionless case, there is no guarantee that a solution exists. Even when a solution does exist, it may not be unique. The first possibility of nonexistence of solution is called *inconsistency* where as nonuniqueness of solution is called *indeterminacy*. Lötstedt [65] realizing that both indeterminacy and inconsistency present major difficulties for a simulation process, has proposed a modification to the Coulomb friction law that eliminates both indeterminacy and inconsistency. Recently Baraff [8] has shown that determining if a given configuration of objects with dynamic friction is inconsistent is NP-complete. Further, for static friction between contacting surfaces, Baraff [8] shows that all one-pont configurations are consistent and speculates that all configurations are consistent. Unfortunately, none of the methods are suitable for use as solution methods in practice as they all require exponential time in the number of contact points.

4.6.3 Penalty Method with Friction

Although the penalty method can be extended to add a tangential friction-like force, it is not clear how or if the complete Coulomb friction model can be accommodated within the framework of the penalty method. In penalty method, a friction is modelled by adding a small amount of viscous drag to the penalty force. The effect of the drag is to resist motion, making the particle come to rest gradually. This also helps to enhance the numerical stability of the differential equation. The resulting equation for penalty force then looks like

$$k_p\,(\boldsymbol{P}(t) - \boldsymbol{X}(t)\,) - k_v\,\dot{\boldsymbol{P}}\,(t)$$

4.7 Remarks

Although the analytical methods are robust and exact, they are more involved and cumbersome to implement as compared to penalty methods. Computationally resting contact is expensive. Penalty method on the other hand makes no distinction between colliding and resting contact. The method is inexact but very easy to implement. A major problem with the penalty method is in choosing the right spring constants. In fact, at times choosing the right spring constant even after many trials is very difficult. As a compromise many practical implementations handle the colliding contact using an analytical method and resting (soft collisions) contact using spring and damper [75].

Chapter 5

Automatic Motion Synthesis by Specifying Motion Features

As humans we perceive all the time, subtle details in the different types of movements of objects and living creatures in nature, and easily distinguish amongst a wide variety of motions ranging from the undulatory crawl of the worm, to the aesthetic walk of a human being. The deep rooted structure underlying motion is so well understood and internalized by us that we are often able to identify the gender of a person just by the style of the walk. Similarly we have no difficulty whatsoever in recognizing the gait of a horse, and are quick to note when it breaks from a trot into a gallop. The variety of movements that we see around us in the world is vast and fascinating. While many of these movements are carried out by living creatures naturally and often unconsciously in a most efficient manner, reproducing similar movements by computer simulation techniques is an extremely complex task.

5.1 The Control Over Movements

As we have seen in the earlier chapter living creatures carry out goal oriented movements by generating internal forces and torques that are complementary to the external forces and torques. These external forces and torques are either present in the environment or are generated in response to their interaction with other objects and creatures in the environment. Animating virtual creatures in a virtual environment thus requires the system to derive the internal forces and torques, formulate the dynamic equations of motion, modify the equations suitably to accommodate any forces and torques generated by interaction with other objects and finally solve these equations of motion to obtain the movement of the virtual creatures. When the forces and torques that cause this movement are to be automatically synthesized, there are basically two methods available to the animator for controlling a virtual creature to move in a fashion as specifically desired by the animator.

- By suitably specifying the performance metric to be optimized.
- By specifying additional constraints which the synthesized motion has to satisfy.

The current implementations of these methods are not very convenient to provide the animator with fine control over the movement, while at the same time ensuring that the animator's efforts in producing the final animations are not excessive. One of the prime problems is due to the fact that generally the method of specifying the performance metric lacks fine resolution capabilities in terms of desirable motion characteristics. For example, minimum energy consumption or maximum distance travelled are typically the goals that many motion synthesis techniques have used. These are at rather too gross a level to be able to automatically result in distinctly different walking movements. The distinction may be subtle but is certainly perceived by us humans.

Constraints do provide a finer level of control. However specifying non conflicting constraints consistent with the goal of optimizing the performance metric is itself a very difficult task. As a result a majority of the methods use key-frames as constraints. Key-frames are again not really simple to specify. For synthesizing even reasonably complex movements of simple articulated figures (with a few DoFs) the number of key-frames required is very large. As the number of key-frames needed increases the animator's efforts increase and correspondingly the animation system's role reduces to one of providing simple interpolation facilities.

In chapter 3 we have discussed a number of optimization methods in which constraints are satisfied by suitably adding constraint satisfaction forces; and that this is one of the prime reasons for the introduction of discontinuities and stiffness in the differential equations of motion. The other methods handle constraints by including them in the objective functions. Such methods do not necessarily guarantee constraint satisfaction and hence may not provide the final results as intended by the animator. In general providing fine control through the specification of additional constraints is neither convenient nor computationally a robust mechanism.

It is therefore not surprising that automatic physically based motion synthesis has not yet been used in any real commercial animation project. Animators continue to use motion capture as the prime technique of synthesizing simulated creature movements. The only other truly convincing simulated movements have been carefully hand tuned by simulation experts after many trials and observations and are for specific movements of specific creatures. It is quite clear that we need motion specification techniques with at least the following properties:

1. In a form that is natural and easy for an animator to specify. It should not require that the animator be initiated into or understand any other discipline like physics.

2. Maximal support should be provided by the system and only the minimally necessary efforts need to be put in by the animator.

3. Progressively fine control over the automatically synthesized motions should be available with the animator.

4. The techniques should not make the dynamic system ill conditioned nor should the computational process of optimization be overly burdened.

5. The technique should be easy to incorporate in an optimization based animation system.

This chapter presents one such motion specification technique – the primary result of our research efforts in this area.

The different movements that we see have distinguishing characteristics, that can very often be easily identified and recognized by humans. Let us refer to these as *motion features*. Two distinctly different movements therefore must have at least one distinctly different feature that enables us to characterize their difference.

Motion features are natural for an animator to specify. A system can then ensure that the optimally chosen simulated motion has all the animator specified features while simultaneously satisfying other constraints required for realistic motion.

Motion features are computed for every simulated trajectory, and therefore their specification does not require any change in the dynamic equations of motion.

Motion features are integrated into the performance metric and when using, say, an evolutionary programming based optimal search technique, they do not in any significant way affect the computational efforts required. On the other hand, if properly specified, motion features can ensure that the optimal solution is reached in fewer generations.

A careful choice of motion features can ensure that fine control is provided both by fine tuning feature values and by an increase in the number of desirable features that the optimal solution should have.

Often very simple algorithms are involved in the computation of features and hence the feature based motion specification technique does not impose any significant additional implementation burden.

Motion control by the specification of motion features thus satisfies all the criteria listed above for a good motion specification technique.

In the rest of this chapter we formally define motion features and carry out a comparative study of feature extraction in other applications like image analysis and computer vision. We then describe the formulation of a performance metric with multiple features. Next we choose a domain of movement types. Specifically we have chosen gaits of legged creatures and describe the formulation and computational procedures for evaluating a number of related features that enable us to easily synthesize different gaits for different types of creatures. This area has been chosen as it is an extensively studied field and we can draw upon a number of reports that give us experimental evidence for the definitions and use of the gait related features [1, 72, 31].

5.2 Motion Features

Motion features are quantifiable attributes that can be used to characterize motion. Once the set of features is specified, given a motion of an articulated figure, it is possible to extract feature values for that motion. All the features need not be present in a given motion. By convention if any feature is not present then it takes the null value. Thus each feature can be considered as a computable function which when applied to a given motion returns a number.

We characterize a motion by a feature vector $\mathbf{f} = (f_1, f_2 \ldots f_n)$ where $f_1, f_2 \ldots f_n$ are the n individual features. In n dimensional feature space, the desired motion will be represented as a point. Motions which have similar features will cluster together in feature space. However, the degree of separability among the different classes of motion will strongly depend on the selected set of features for an application. Given a feature vector value, the task of the motion synthesizer is to search for the motion having its feature vector identical or close to the given feature vector.

Feature extraction is a very well studied subject in pattern recognition and computer vision [84, 7]. In pattern recognition for example, features of known classes of pictures are computed and distinct clusters of features are defined in feature space, each cluster representing a distinct class of pictures. Given any unclassified picture, its features are computed and then matched with the predefined set of feature clusters and classified according to its proximity to the clusters. The primary emphasis is on the development of efficient and robust methods for clustering and classification. For historical reasons, pattern recognition and image processing work are usually closely tied together and most often the pictures are raster images and features are derived using a variety of image processing and analysis techniques.

Another approach to pattern recognition is known as model based or structural pattern recognition [18]. In this approach a skeletal structure of the image is computed and then classified. The elements of the skeletal structure and their connectivity mechanism are themselves used like features for classification.

While there are a few exceptions [46, 97], pattern recognition so far has primarily been a static picture recognition process. If moving images are used, like in some computer vision studies related to vehicular movement [43], then they are largely used for producing additional information for hypothesis validation or filling in of missing information in the static scene.

Another major area in computer vision is to study moving pictures for synthesizing shape information. The idea is to reconstruct the surface geometry of an object from a sequence of images of the object as seen from a camera in relative movement with respect to the object [43]. A primary motion feature in most of the "shape from motion studies" is what is termed as optical flow. Very simply stated, optical flow is the rate at which a pixel of an image changes its intensity. The optical flow feature values are then used to synthesize the shape (by surface curvature) around the object surface region corresponding to this pixel [41].

There are a number of fundamental differences between features as used in pattern recognition or computer vision studies and features as used by us in motion synthesis.

1. While newer sensing mechanisms like the range sensing devices [21] are becoming available and making three dimensional geometric data about dynamic scenes available for use in pattern recognition and computer vision, most of the emphasis in these fields has been and continues to be on colour/intensity images. On the other hand in our approach the features

are those which can be easily specified by the animators and therefore are necessarily not pixel image features but features of the actual motion in 3D – features that are completely independent of the final rendered image of the animation sequence.

2. Just because of the sheer volume of image data involved in a sequence, feature extraction from sequences of moving pictures with the intent of classifying different types of motion has not been a significant area of study in pattern recognition or computer vision. Our approach primarily depends on features of movements.

3. Since the primary purpose of features obtained from feature extraction in pattern recognition and computer vision is for use in other computer processes like classifying or surface reconstruction, it is possible to use low-level pixel based features of pictures. And most methods do define automatic procedures for pixel level feature extraction. Our approach requires motion features at a much higher level. They are to be specified by the animator and then used for motion synthesis.

   The specification of high level features and their use for synthesis is an approach that has been proposed for computer aided geometric design also. Feature based design of solid geometric models is a popular and promising topic of research and development in the field of geometric design. There is once again a major difference in strategy. In feature based geometric design, features are more like parameterized macro component specifications, which are used in a deterministic closed form fashion to compose the object [83]. Considerable complexity is introduced due to the fact that during the process of composition the resulting object has to be checked for validity in all its intermediate stages. Unlike in our approach where we synthesize a motion and then analyze its features, there is no analysis of features as such, other than validity checks. Also we are not aware of any global optimization based approach to geometric design by features.

In our method of synthesizing motion, we have to choose that motion which best matches the animator specified features. Since these features are to be extracted from a completed motion, it is mandatory for us to simulate the motions and carry out the feature extraction for each of the motions. There are three important aspects that play a significant role in the success of our method.

- The set of features that are to be extracted for each simulated motion.
- The motion representation space which is explored when searching for the optimal motion with the best of the given features.
- The method of matching the simulated motion features with the desirable set of features and obtaining a measure of closeness of the motion to the desired one. This is essentially the performance metric.

5.3 Features

There is no limit to the number of features that can be defined for a movement. The set of features that have to be chosen for the synthesis of different classes of motions is therefore not a simple problem to solve. There are a few conditions that the chosen set of features should satisfy which make this problem difficult and these are discussed below.

1. **Computability:** Every feature in the set of features chosen must be computable for every trajectory that the computer is capable of generating for the given figure.
2. **Discrimination:** The chosen set of features must clearly map different classes of motion of the figure into distinctly separate clusters in the feature space.
3. **Describability:** The chosen set of features should be rich enough and at a significantly high enough level so that the different classes of motion and different movements within each class of motion can be intuitively

described by the animator by specifying desirable features of the movement.

4. **Minimality :** The chosen set of features should have a minimal size so that the dimensionality of the feature space is reduced and overall computational efficiency is increased.

5. **Robustness in control:** The chosen set of features should not be overly sensitive to fine changes in control parameters. That is, a small change in control parameters should not give rise to motion with completely different features.

From the above it should be clear that there is no universal set of features that can be used for all classes of motions. The set of motion features must be chosen in a domain specific manner. It is also clear that identifying what features should go into the set would be highly dependent on extensive studies of real life movements of that domain. As already mentioned for our experimentation we have chosen the domain as the movements (gaits) of legged creatures. Fortunately for us, as we shall soon see, this is an area that is very rich in movement types, is widely applicable and has also been studied extensively [1, 72, 86].

5.4 The motion control representation space

In principle the motion synthesis method with features could be based on any of the motion representation schemes discussed in earlier chapters. All that is needed is to be able to simulate the motions and then analyze them for the chosen set of features. The optimization technique used determines how the different motions are explored in the motion control space. In our implementation we have chosen to use an evolutionary based global optimum search algorithm. This algorithm explores the control space in discrete steps starting simultaneously at a number of points. The entire procedure will be more efficient if the dimensionality of this control space is kept low. We have therefore chosen to represent motion through motion controllers. Specifically we have resorted to the use of pose control graph controllers, because they are the simplest.

However other motion controller representations like the banked stimulus response controller or sensor actuator network controller could easily have been used without any significant difference in our implementational efforts or to the computational resources needed.

5.5 Feature Based Performance Metric

Let X denote the space of motion control representations and x denote a specific motion controller.

Let $f(x) = f_1(x),\ f_2(x) \ldots f_n(x)$ denote the feature vector with n separate feature components. $f_1(x)$ to $f_n(x)$ are computed for the motion resulting from the execution of the controller x. Without loss of generality we shall assume that all $f_i(x)$ are normalized, *i.e,* $0 \leq f_i(x) \leq 1$ and $f_i(x) = 0$ implies that the feature is not present in the motion.

If $y = y_1, y_2 \ldots y_n$ is the desired feature vector as specified by the animator then a performance metric can be defined as follows:

$$\min_{x \epsilon X} \{|f(x) - y|\}$$

That is the distance of the motion mapped as a point in the feature space from the desired feature space point provides us with a measure of the extent to which simulated motion deviates from the motion with desirable features. Smaller this distance closer the match. It is possible to weight the importance of individual features, with these weights being under the control of the animator. The performance metric can then be reformulated as follows:
Let $w = (w_1, w_2 \ldots w_n)$, all $w_i \geq 0$ denote the weight vector. Each feature deviation can then be weighted and denoted by $d_i(x)$, with $d_i(x) = w_i(f_i(x) - y_i)$. The performance metric is defined as:

$$\min_{x \epsilon X} \{|d_i(x)|\}$$

Throughout the rest of this thesis and in our implementation we shall assume that this metric is used during optimization.

5.6 The Gait of Legged Creatures

The study of motion like walking, running *etc.* has been a subject of fascination for many. There have been many attempts to classify different types of motion among quadrupeds and bipeds [86]. One of the generally recognized features of natural legged locomotion is that animals typically employ their limbs in a number of distinct periodic modes. Thus we say that a man walks, runs or leaps and a horse trots, canters, gallops, *i.e* such modes are identified by characteristic patterns of foot falls ordinarily called gaits [45]. For example, in bipeds, walking and running can be distinguished from a mechanical point point of view on the basis of a simple test – in running, but not in walking, there is a period when both feet leave the ground . The synthesis of realistic walking or running gaits is however a very difficult problem that has been the focus of a large body of research in computer animation [35, 87, 71], biomechanics [106], computational neuroethology [12] and robotics [22].

In computer animation for example, in recent years we have seen an increasing number of films in which legged virtual creatures move in a realistic fashion. The computer animation techniques used are however relatively simple. Motions of mechanically moving toys or living humans are captured from film or video (rotoscoping) or by pasting electronically tractable sensors directly on the moving figures. The captured data is used with some minimal editing for creating similar animated movement of a very similar virtual figure. This technique is simple and gives excellent results as it is almost like playing back of real phenomena. There are however several limitations. It is best suited for animating human movements. It is difficult to ensure that other living creatures move as desired by the animator. Motion captured from mechanically moving toys will most often will not look realistic. It is also difficult to make mechanical figures move with the kind of variety that live creatures move. Furthermore, once a motion is captured it is difficult to change or reuse it even if it only requires a slightly different situation.

The specific challenge of creating tools for human animation has been taken on by a few others also in computer animation. The work of Badler *et al.* [5], Thalmann and Thalmann [79], and Boulic *et al.* [13] illustrate the present state of the art in modelling humans using predominantly kinematic methods. In another method a technique for generalizing existing rotoscope data while preserving original motion characteristics is presented [59].

Beyond kinematic methods, there have been several proposals to produce animated walking motions using physically-based models. Girad [33] uses a mix of kinematic and dynamic methods to achieve a variety of biped and quadruped motions. Bruderlin and Calvert [16] use a similar mix of techniques to generate realistic parameterized walking motions for a kinematically complex human model, and later show that parameterized walks can also be achieved using a purely kinematic model. The gaits are constructed using control rules extracted from experimental gait data.

The work of Raibert and Hodgins [86] demonstrates an elegant and robust control solution for balanced hopping and running creatures having one, two, or four legs. Hodgins *et al.* [49] have also developed a variety of control algorithms for tasks such as running, diving and bicycling. Stewart and Cremer [95] use changing sets of desired constraints to control the motion of a human-like model in climbing and descending stairs. McGeer [69] shows that stable passive walks can be achieved down modest inclines. McKenna and Zeltzer [71] show how to synthesize a variety of gaits for a fully-dynamic hexapodal model. There is a significant body of robotics and biomechanics research concerning the control of bipedal walking motions, as well as for simulating human motion. While specific control solutions abound, there has been relatively little work on the automatic synthesis of gaits in a general setting, *i.e*, for creatures of arbitrary design.

Kelso and Pandya [56] have studied the order and regularity exhibited in human and animal motion and viewed the various gaits in terms of phase dynamics. The theory originated from studies of human movement coordination in rhythmic bimanual tasks. It is observed that when the human right hand is involved in a task which is out of phase with the left hand, there is a spontaneous switch of coordination at certain movement frequency. This shift in synthesis of different gaits of a creature. Interestingly, despite the difference in morphology, at intermediate and high speeds, two, four, six and eight-legged animals produce ground force patterns that are fundamentally similar. All can run, or bounce. Running humans, trotting dogs, *etc.* can move their bodies by producing alternating propulsive forces. Two legs in a trotting quadrupedal mammal, three legs in an insect, and four legs in a crab can act the same as one leg does in a biped during contact. The centre of mass of the animal undergoes repeated acceleration and deceleration with each step, even when travelling at constant average velocity. Motion analysis studies of walking and running of bipeds have shown that changes in the potential and forward kinetic energies of the centre of mass are almost exactly out of phase in walking so that the total energy changes only a very little throughout walking steps. The opposite is true for running, where changes in potential and kinetic energy are substantially in phase leading to a large change in total energy (cf Figure 5.7).

Further evidence of this equivalence comes from the the dynamic similarity principle proposed by Alexander [1]. His hypothesis is based on the fact that despite of the difference in physical structure, many natural motions show remarkable similarity. For example, two pendulums of different lengths swinging through the same angle have dynamically similar motion. The dynamic similarity hypothesis predicts that animals of different sizes tend to move in dynamically similar fashion within their physical limits. It is obvious that animals cannot move in precisely dynamically similar fashion unless their bodies have similar physical structure. However, comparisons of related animals of grossly different sizes show remarkably little deviation from dynamic similarity.

All the above studies show that there does exist a set of simple features in different gaits which can be used to distinguish amongst them. These features are at a high enough level and can also be specified by animators as desirable features which the synthesized motion must have. We discuss next the set of features used by us for synthesizing different gaits for a number of different virtual creatures, both one legged and two legged.

5.7 Gait Related Features

Our interest is in gait patterns that repeat themselves. The sustained gaits of animals are all nearly like this. The gait patterns of an articulated figure can be characterized using a number of terms. These terms are directly usable as features at times and in other cases they would be used to compute feature values. We describe the various terms used in gaits.

Gait cycle: Each repetition of the gait pattern is called as the *gait cycle* and the duration to complete one gait cycle is called as the *gait period* (cf Figure 5.2). Gait period is a feature which differs for different types of movements.

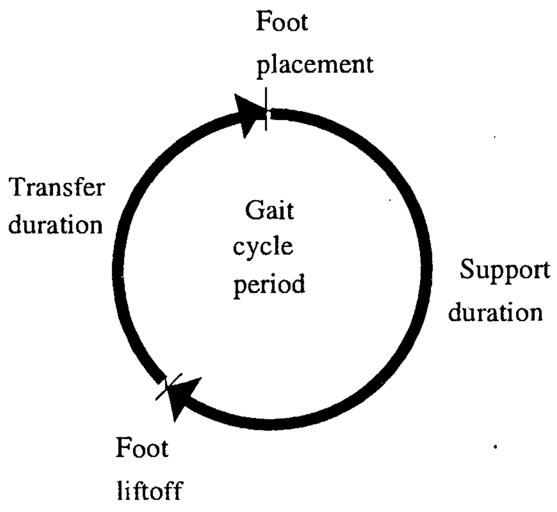


Figure 5.2: Gait cycle

Step length: This is distance travelled by the articulated figure in one gait cycle, for example, from the setting down of a particular foot to the next setting down of the same foot (cf Figure 5.3). Step length is again a feature which differs for different gaits.

Duty factor: This is the percentage of time spent by any given leg on the ground. The duty factor is an excellent feature to use for distinguishing between walking and running movement of bipeds. The distinction generally made is that walks have duty factors greater than 0.5, so that there must be stages in the gait cycle when both feet are on the ground simultaneously. Similarly runs

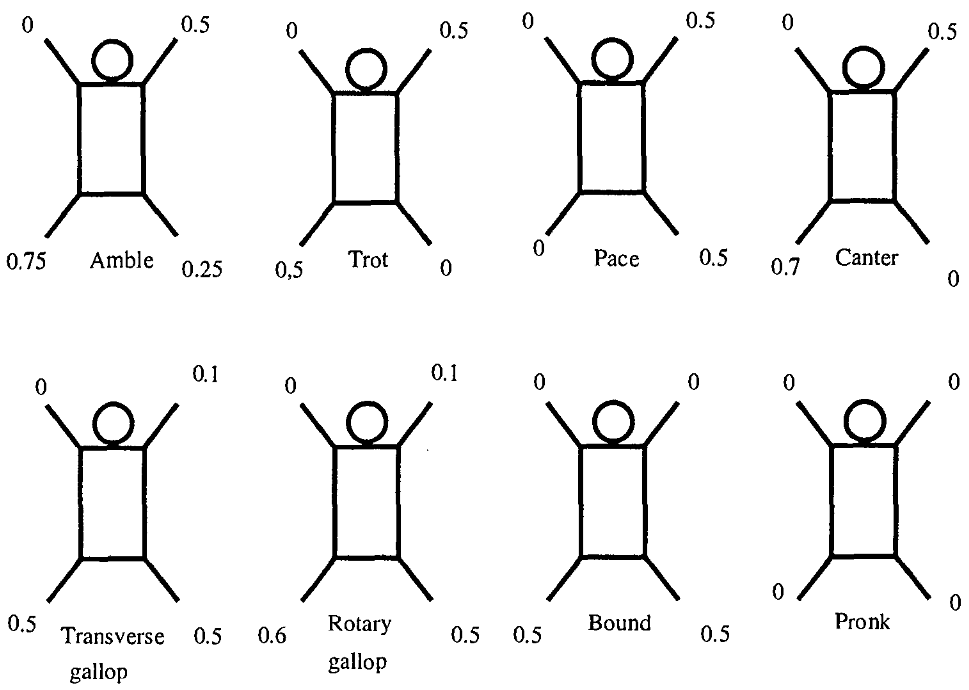


Figure 5.5: Diagram showing relative phase amongst legs of a quadruped

$$G = \begin{bmatrix} 0\,0\,0\,1 \\ 0\,1\,0\,1 \\ 0\,1\,0\,0 \\ 0\,1\,1\,0 \\ 0\,0\,1\,0 \\ 1\,0\,1\,0 \\ 1\,0\,0\,0 \\ 1\,0\,0\,1 \end{bmatrix}$$

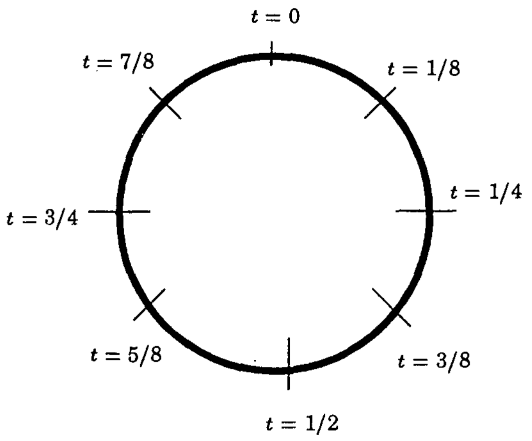


Figure 5.6: Gait matrix and corresponding phase sequence for a quadruped walk

Centre of mass: This is the point were the entire mass of the body appears to be concentrated. For an articulated body it can be computed as follows:

Let us denote the length of the ith link by L_i, mass by M_i, the angle of the leg with respect to the horizontal line by θ_1 and other internal angles as $\Theta = (\theta_2, \theta_3..\theta_n)$. Let (X_i, Y_i) denote the centre of mass of the ith link and (X, Y) the centre of mass of the articulated figure.

$$X_i = \sum_{k=1}^{i-1} L_k \cos(\sum_{j=1}^{k} \theta_j) + 0.5\ L_i\ \cos(\sum_{j=1}^{i} \theta_j),$$

$$Y_i = \sum_{k=1}^{i-1} L_k \sin(\sum_{j=1}^{k} \theta_j) + 0.5\ L_i\ \sin(\sum_{j=1}^{i} \theta_j),$$

and

$$X = \frac{\sum_{i=1}^{n} M_i X_i}{\sum_{i=1}^{n} M_i},$$

$$Y = \frac{\sum_{i=1}^{n} M_i Y_i}{\sum_{i=1}^{n} M_i}.$$

The variation over time of the position of the centre of mass is an oft used feature for differentiating amongst different types of movements.

Total energy: The total mechanical energy of the centre of mass of the body during walking and running is very different (cf Figure 5.7). Total energy consists of kinetic energy $E_k = \frac{1}{2}mv^2$, where v is the speed and gravitational potential energy $E_p = mgh$, where h is the height and g is the gravitational acceleration of the centre of mass.

Froude number: The dynamic similarity hypothesis predicts that animals of different sizes will use the same gait when traveling the same Froude number. The Froude number is defined as a ratio u^2/gh where u is a speed characteristic

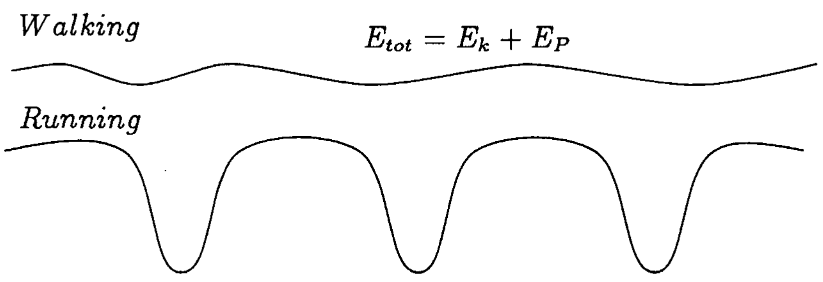


Figure 5.7: Variation in total energy while walking and running

of motion, g is the acceleration due to gravity; and h is a characteristic length of the leg.

Table 5.1 reproduced from [31] shows the Froude numbers and a few other features for a variety of legged locomotion.

Creature	*No. of legs*	*Hip height* $h\ (m)$	*Speed* $u\ (m/s)$	*Frequency* $f\ (Hz)$	*Froude no.* $u^2/(gh)$
Crab walking	8	0.035	0.4	3.2	0.4
Man walking	2	0.9	1.6	1	0.3
Dog walking	4	0.5	1.3	1.6	0.4
Crab trotting	8	0.035	0.9	6.2	2.4
Cockroach trotting	6	0.004	0.3	13	1.7
Man jogging	2	0.9	3.3	1.6	1.2
Dog trotting	4	0.5	2.7	2.2	1.5
Turtle	4	0.07	0.1	0.6	0.02

Table 5.1: Comparison of individual leg dynamics

5.8 The Use of Gait Related Features

We have incorporated gait related features in our global optimal search technique. Just by specifying different values for some of these features we have been able to synthesize different kinds of gaits for virtual creatures with different number of limbs. The implementation and the experimentation results are discussed in detail in the next chapter.

Chapter 6

Implementation and Results

In order to be able to carry out some experiments on motion synthesis through the specification of motion features, it is essential to have a physically based motion simulation environment. A considerable part of the research efforts reported here have been towards the creation and implementation of such a simulation environment. This chapter presents the details of our implementation of this simulation environment and also reports a number of results from various experiments in motion synthesis carried out using this implementation. As we shall see, the simulation environment is powerful and yet simple and will enable a variety of motion simulation related experimental research and development to be carried out.

In our implementation the entire process of motion synthesis is divided into three phases (cf Figure 6.1). In the first phase an optimal controller is synthesized using the stochastic population hill climbing algorithm. In the second phase the motion is recorded frame by frame by executing the controller and simulating the motion. In the third phase the recorded motion is played back.

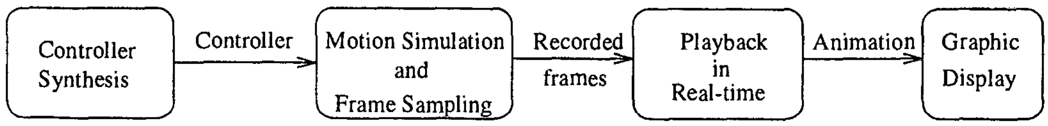


Figure 6.1: Three phases in motion synthesis

In what follows we shall explain in detail each of these phases and also illustrate our method with a few representative examples of movements of articulated figures. Since limited computational resources were available for this work, we have made the following simplifying assumptions:

- The system simulates movements restricted to two dimensions.
- Only tree structured articulated figures are considered.
- The only external interacting object modelled is the ground.
- Links of the articulated figures are connected using joints of a single type, namely pin joints.
- In a single simulation we consider the motion of a single articulated figure.

It is important for us to emphasize here that the simplifications and assumptions stated above are not in any way inherent limitations of our approach. The implementation is highly modular, and is flexible enough to easily incorporate extensions so that movements of multiple 3D articulated figures with different kinds of joints in a complex 3D virtual environment can be handled.

6.1 Controller Synthesis

The different components of the controller synthesis process are shown in Figure 6.2. The animator specifies the physical and the geometric structure of the articulated figure and with motion features the movement task to be performed. Once the motion features are defined the fitness function is composed and the search is undertaken using stochastic population hill climbing algorithm for a suitable controller. The output of the controller synthesis stage is a controller with values assigned to all its parameters. We shall now explain each of the components and their functions.

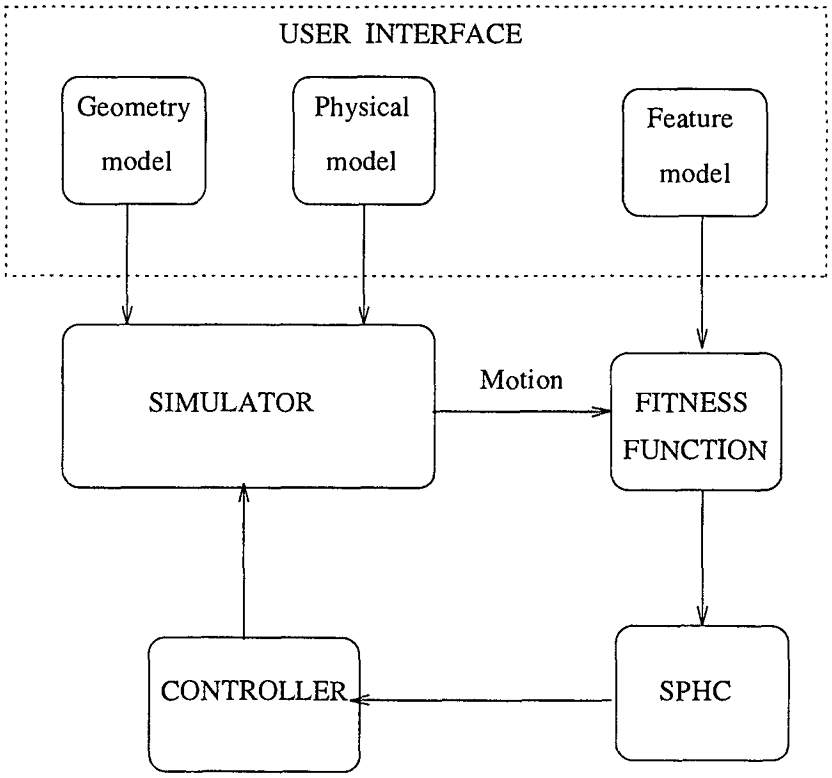


Figure 6.2: Different components in controller synthesis phase

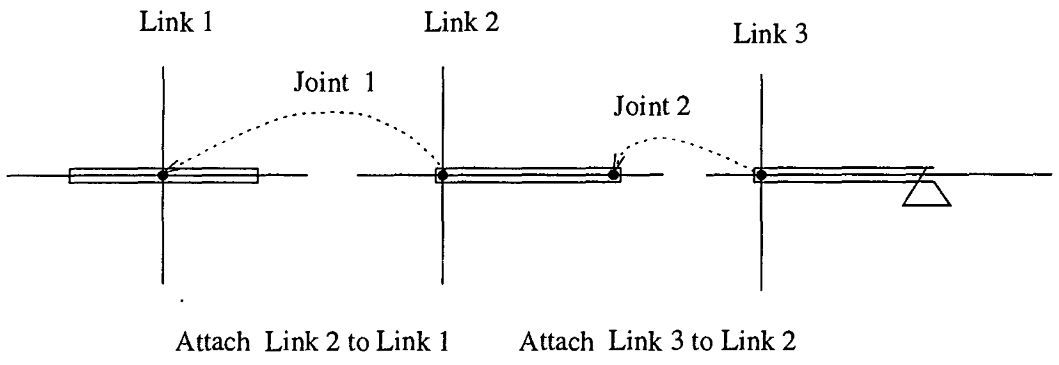


Figure 6.3: Geometric structure specification

6.1.1 Geometry Model

The geometry model stores all the relevant geometric information about the articulated figures and the environmental objects. The system allows the animator to describe the geometric structure of an articulated figure. Each link is specified with respect to the coordinate system attached to it (cf Figure 6.3). The specification includes, the length of each link and its relation with other links.

Geometric Description Script

A link type is defined as:

```
objname <name_of_the_object>
path
 pts  x1 y1
 pts  x2 y2
  .    .  .
```

Once an object of this type is defined, many instances of this object type can be created where each object instance is a link as follows:

```
instance <name_of_the_object> <link_name>
```

Attachment of two links can be indicated to the system by the use of:

```
attach <link_name> to <link_name>
```

Local (body) transformations (translation, rotation etc.) on the link can be done using:

```
tf trans <link_name> x y
```

Environmental object geometry is also part of the geometry model. As already mentioned we have chosen to model only ground interactions. Hence we accept position and orientation of the ground plane as specified by the animator. The plane of the ground can be edited to change its orientation. In particular, the orientation of the normal vector can be adjusted. This allows sloped ground planes to be tested with the articulated objects.

6.1.2 Physical Model

The physical model stores animator specified physical properties of each individual link such as mass, inertia, position of the centre of mass etc. Although for visualization purposes links are modelled as two dimensional entities, physically they are modelled as one dimensional rigid link-segments. This simplifies the computation to a large extent. For example, the inertia tensor gets transformed from a 3×3 matrix into a scalar. Typically the inertia is specified in the coordinate system attached to the centre of mass. Each link has a unique parent link and one or more child links. Further, for each articulated figure there is a special link called as the root link. The root link is attached to the observer's frame of reference. Each child link is attached to the parent link at the origin of the child link. By convention the parent of the root link is assumed to be 0 (cf Figure 6.3). In order to detect collision between the articulated figure and the ground, several points on the body of the articulated figure are identified. These points are called *monitor points*. Monitor points are continuously monitored during the course of simulation. Whenever any of the monitor points are found to collide with the floor, an appropriate response force is computed by the simulator module and applied at the monitor point.

Physical Description Script

A link is defined as:

```
link <link_num> <parent_num> <attach_x> <attach_y> <mass> <inertia>
<cmass_x> <cmass_y>
```

where, <link_num> is a number given to this link in the articulated body.
<parent_num> is the number of the parent link to which this link is connected.
Root link always has <parent_num> as 0.
<attach_x>, and <attach_y> are the coordinates of the point on the parent link where this link is connected to it.
<cmass_x> and <cmass_y> represent the center of mass of the link.

If the link is fixed at some place say ground or roof it is specified as:

```
fix <link_num> <fix_x> <fix_y>
```

where <fix_x> and <fix_y> are the coordinates of the fixed point on this link.

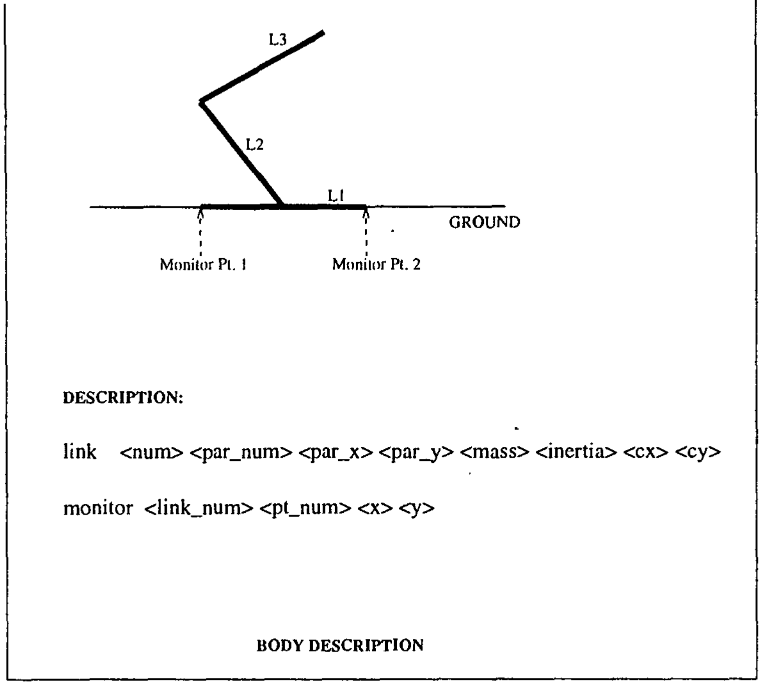
Monitor points are used to monitor the points on the body which are expected to collide with the ground (cf Figure 6.4). These are defined as:

```
monitor <link_num> <monitor_pt_num> <pt_x> <pt_y>
```

6.1.3 Feature model

The feature model stores feature values specified by the animator. Each feature is specified by a key word followed by a value and the allowable range for that feature. For example:

```
feature <value> <min> <max>
```



DESCRIPTION:

link <num> <par_num> <par_x> <par_y> <mass> <inertia> <cx> <cy>

monitor <link_num> <pt_num> <x> <y>

BODY DESCRIPTION

Figure 6.4: Articulated figure with monitor points

6.1.4 The Simulator

In addition to the articulated figure's geometric information the user can directly set parameters for controlling different aspects of simulated environment such as simulation time, magnitude and direction of gravity. Gravity control allows the direction and magnitude of the gravitational acceleration vector to be modified. By setting the components of the gravity vector to zero, gravity can be effectively turned off.

The equations of motion for our articulated bodies are too complex to derive by hand. In addition we wish to have an efficient implementation, since these equations need to be solved at every time step. We were able to integrate into our simulation environment "Dynacomp" [99] a public domain dynamics compiler that symbolically computes equations of motion in the form $Ax = b$ given a physical description of the articulated figure as input. The output of the dynamics compiler gives the symbolic value for each element of matrix A and

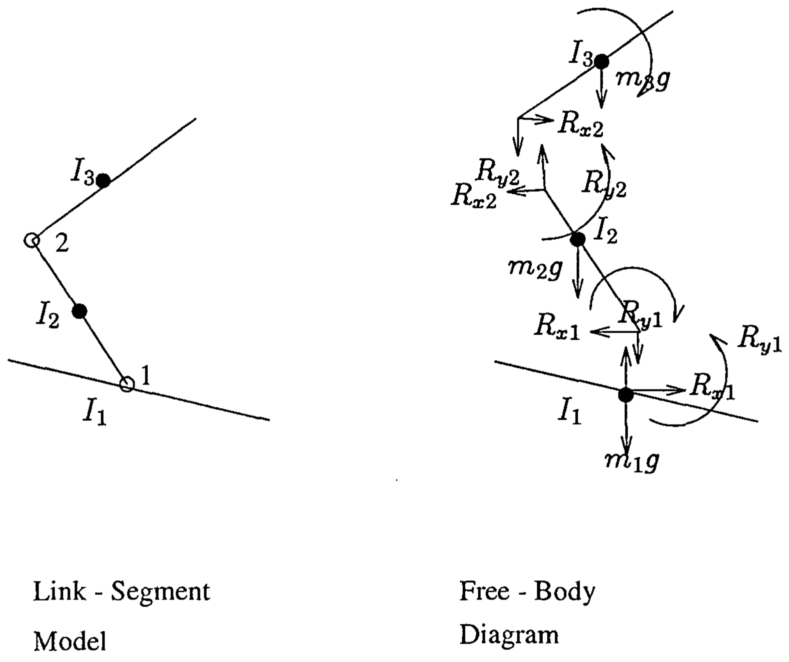


Figure 6.5: A relation between free-body diagram and the link-segment

also to the vector b. Values of common subexpressions are precalculated to avoid unnecessary calculations. The implementation uses the recursive Newton-Euler formulation (See section 3.3.2). This is $O(n^3)$ in the number of links and is quite suitable for $n < 10$. The values of A and b are output as lines of "C" code so that the equations of motion can be compiled. The LU decomposition method is used to solve the linear system of equations for the accelerations. A and b are dependent on the internal torques applied at the joints, external forces, the physical properties of the links, and the state of the links. x represents the vector of unknown accelerations. The accelerations are numerically integrated using the simple Euler method to determine new velocity and position of the links. The time step is chosen to be in the range of 0.001 to 0.0005 in order to overcome the stiffness problem.

In our model, the creatures are treated as free bodies in space (cf Figure 6.5). That is all the forces and torques acting on an individual segment are added up to compute the total force and torque acting on the segment. Apart from the

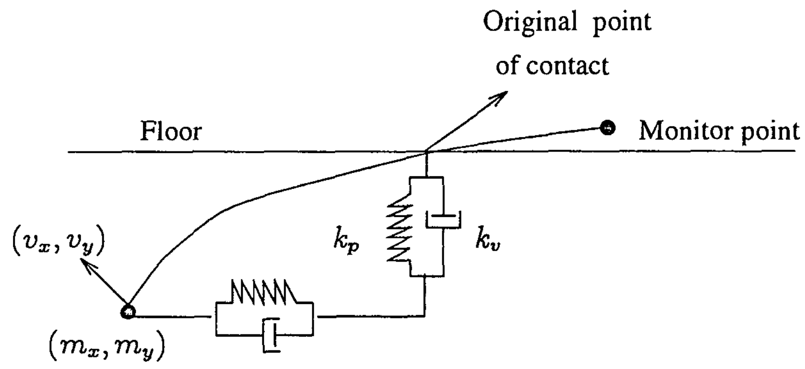


Figure 6.6: Modelling of the floor

internal forces acting at the specified key joints, the only other external forces modelled are the gravitational force and the reaction force exerted by the floor. The acceleration of each individual link-segment is computed by calculating all forces/torques acting on the segment. The forces exerted by the floor on the articulated figure are calculated using stiff spring and damper model. We favour this approach as it is simple and flexible and also the same formulation of the equations of motion can be used throughout the entire simulation. There is no need to model and compute the magnitude of the impulsive forces that occur upon impact with the ground. The external forces exerted on the figure are computed at the points of contact with the floor which are typically the monitor points (cf Figure 6.6). The position and velocity of these monitor points on the articulated figure are used to compute the external forces as follows:

$$F_x = -(m_x - p_x)k_p - v_x k_v$$

$$F_y = -(m_y - p_y)k_p - v_y k_v$$

where (m_x, m_y) is the present position of the montior point and (p_x, p_y) is the point of initial contact with the floor. Typically spring and damper constants chosen are $k_p = 10^5 N/m$ and $k_v = 10^3 N/m$. This creates a suitably stiff floor that functions effectively when used in a simulator with a sufficiently small time step. A simulation script is used to define the various parameters used in simulation.

Simulation Script

The time step used in simulation is set using

```
set dtsim <step_size>
```

The initial state the articulated figure is set using:

```
state <x> <dx> <y> <dy> <th1> <dth1> <th2> <dth2> ....
```

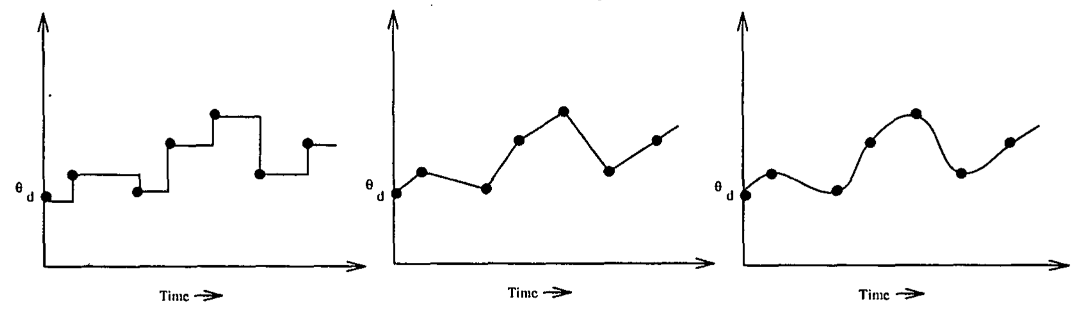
where	<x>	– is the x position of the origin of the root link
	<dx>	– x speed
	<y>	– y position
	<dy>	– y speed
	<th1>	– th1 angle of the link 1
	<dth1>	– dth1 angular speed of link 1
	...	

Simulation time can be set by:

```
sim <time_for_simulation>
```

6.1.5 Controller representation

The choice of representation for the solutions plays a crucial role in the success of the evolutionary algorithm. Ideally, the representation should be compact enough so that the motion synthesis problem can be solved in a reasonable time, without sacrificing generality. Compactness is achieved by having fairly powerful rule based controller representations that need a small number of states and hence a small number of parameters. In our implementation, we have used the pose control graph and PD control law given by the equation $\tau = k_p(\theta_d - \theta) - k_v\,\dot{\theta}$ to represent a controller. Where k_p is the spring constant, k_v is the damper constant, θ and $\dot{\theta}$ are the current angle and the angular velocity respectively and θ_d is the rest angle (equilibrium position) of the spring.



(a) (b) (c)

Figure 6.7: (a) Piece-wise constant (b) Piece-wise linear (c) Continuous

The advantage of this controller is that the torque function gets automatically defined once the target joint angle is specified. To execute a particular movement of the joint, it is necessary to define a number of target joint angles. The motion synthesis problem is then converted to that of synthesizing the function $\theta_d(t)$. If the articulated figure has m actuators, it amounts to synthesizing m functions of the type $\Theta_d(t) = (\theta_d^1(t), \theta_d^2(t) \ldots \theta_d^m(t))$. The simplest way to solve the problem is to choose a piece-wise continuous function. This function could be a constant, could be linearly varying or could be more complex with continuous basis functions such as splines [20], sinusoids or wavelets [64] (cf Figure 6.7). For the sake of simplicity, for our experimentation purposes, we have used piece-wise constant functions.

To synthesize a motion sequence for duration T, we divide T into several phases or states. Each phase will be associated with a set of parameters such as

$$-, \theta_d, k_p, k_v$$

where

i — i^{th} joint actuator

θ_d^i — desired angle for the i^{th} joint

k_p^i — spring constant corresponding to the i^{th} joint

k_v^i — damping constant corresponding to the i^{th} joint

t — time duration of the phase.

If there are fifteen phases in a motion sequence, the number of unknowns to be determined are $15 \times 4 = 60$. We have mentioned earlier, that, as the simulation duration increases, the search time increases exponentially. When we consider periodic motion, such as walking, running, hopping etc., one can reduce the search time. In periodic motion, after every period t_{per}, the motion is repeated to fill the simulation time T. Unknown parameters depend only on the number of phases in the period t_{per} (cf Figure 6.8). In order to reduce the search time

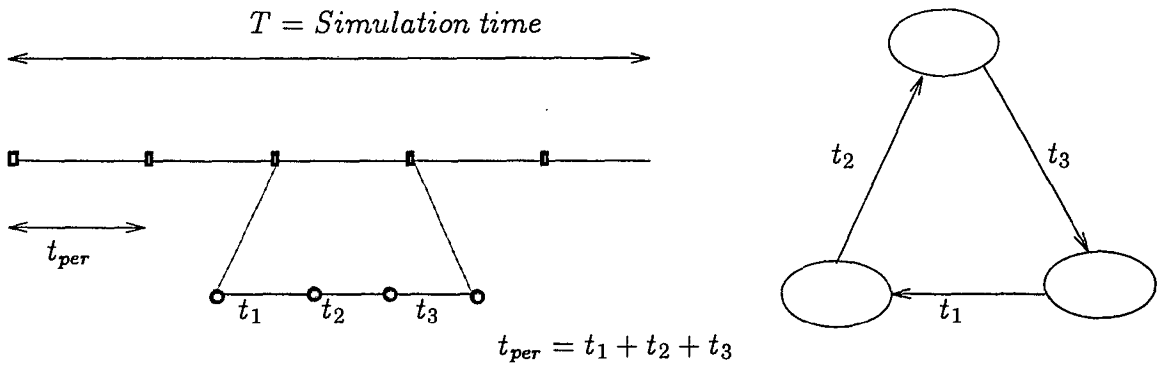


Figure 6.8: An illustration of the controller

further, one can fix the values t, k_p^i, k_v^i a priori. The time period t could be either derived from a previously synthesized key-framed version of the motion or from live or video data of a creature similar in size and shape [103]. Values of spring constants can be estimated depending on the mass of the body. For heavier links, higher values of spring constants are required in order to protect the springs from a possible collapse (spring failure). However, values should not be so high that they would generate such high torques at the joints that unexpected motions are caused. For more details refer to Appendix A.

6.1.6 Performance Metric

For our experimental purpose, we have considered only a few features to characterize the motion. Although this simplifies the implementation considerably, it fully retains the essence of our methodology. Our task is to synthesize periodic motions such as hopping, running and walking. Some of the features that we have built into our implementation and experimented with are :

- external energy (E)
- horizontal distance travelled by the centre of mass (D)
- intermediate postures ($\theta_{s,i}$), fully or partially specified

The specific performance metric is as follows:

$$f = w_1 * \sum_{s=1}^{samples} \sum_{j=1}^{joints} (1 - (\theta^o_{(s,j)} - \theta_{(s,j)})) + w_2 * (1 - \frac{(E_o - E)}{E_{max}}) + w_3 * (1 - \frac{(D_o - D)}{D_{max}})$$

where,

w_1, w_2, w_3	are weights, assigned to the features depending on their relative importance, the value ranging between 0 and 1.
$\theta_{(s,j)}$	is the angle at joint j for posture,
E	is the external energy
D	is the horizontal distance travelled
E_{max}, D_{max}	are the maximum expected external energy and horizontal distance, respectively. These values are used for normalization of the two quantities.
$\theta^o_{(s,j)}, E_o, D_o$	are the feature values specified by the animator.

For more intuitive explanation of various terms used in the performance metirc, refer to Appendix A.

6.2 Stochastic Population Hill Climbing Algorithm (SPHC)

The SPHC algorithm is an evolutionary programming algorithm [30] that can be distinguished from genetic algorithms, primarily by the fact that it uses only the mutation operator and does not use a crossover operator.

Like all genetic algorithms SPHC uses a population of solutions. Each solution in the population is perturbed randomly at each iteration, using the mutation operation with probability 1. The resulting solutions are compared with their original solutions and the better ones are kept for the next generation. Periodically, a reseeding operator is applied which selects the top half of the population and copies them into the bottom half of the population, refocusing the search on the most promising solutions in the population. The full algorithm is shown in Figure 6.9

```
Initialize population
Evaluate each solution in the population
for generation = 1 to number_of_generations
  for each individual solution in the population
    Randomly perturb the solution
    Evaluate the new solution
    if the new solution is better than the old one then
      Replace the old solution with the new one
  end for
  if(generation mod reseed_interval) = 0 then
    Rank order the population
    Replace bottom 50% of the population with top 50%
  end if
end for
```

Figure 6.9: Stochastic population hill climbing (SPHC)

Mutation Operator

The mutation operation is the backbone of our SPHC algorithm. In every iteration all solutions go through this operation. Since a slight change in parameters can change the motion drastically, it is necessary to apply the mutation operation with care. We have selected to mutate only one parameter at a time with only a small change in original value. The parameter to mutate is selected randomly with all the parameters having equal probability. This was found quite suitable through experiments, as it helps the algorithm to fully explore the region near to the existing solution. If we try to mutate more than one parameter at a time, the solution may jump from one region to another without exploring the current one. As the function is multimodal, it may actually be the case that the optimal solution is in the vicinity of solution being mutated. Each time mutation is called either the selected parameter goes through a *creep* operation[1] or all its parameters are randomized from scratch (cf Figure 6.10).

```
Randomly select one of the states in pose-control graph to be modified
Randomly select operation to be applied on selected state
IF  operation is  creep operation   THEN
    Randomly select one of the creep operations and apply
ELSE
    Generate all the parameters in the new state randomly from scratch
```

Figure 6.10: Mutation operation

6.2.1 A Parallel SPHC

Controller synthesis is computationally a very expensive process. The reasons are two fold. Firstly, the time taken for a single simulation is large. For example, an eight second simulation took around two minutes on a VAX 8600 machine. The simulation time is directly proportional to the complexity of the creature i.e number of links and the monitor points. Secondly, due to the fact that search space is very large, the search algorithm has to make many

[1]The creep operation is used here to modify the parameter by a very small factor. For more details refer to Appendix A.

simulations before locating a suitable controller. The overall time taken by the synthesis process can be reduced considerably if we parallelize the search task. In SPHC algorithm we have a population of solutions which are to be modified and checked separately. All the evaluations and mutations are independent. We can take advantage of this independence property in parallelizing the algorithm.

In the best case, if we have the number of processors equal to the population size, we can run all the simulations separately on each of the processors, achieving maximum parallelism at the granularity of a single simulation. If the processors are lesser in number than the population size, a good scheduling policy has to be implemented to achieve considerable amount of parallelism. Since simulation time for each candidate solution in the population is the same, it is easy to parallelize the search process.

We have implemented the algorithm on a networked environment. There are several heterogeneous workstations connected on the network, each having a different load average at any time instance. Also there are varying communication delays on the network. In addition to parallelizing the code to distribute the evaluations on different processors, we have to handle the problem arising due to varying communication delays. We have adopted a very simple solution to the problem. We treat the population of controllers as one common pool of tasks which are allocated to the set of processors. To start with, all the processors are allocated one candidate each for evaluation. As soon as any of the processors becomes free, it is allotted a new candidate.

To handle the parallelization problems in the network environment we have made use of a system called Parallel Virtual Machine (PVM) [27].

Application programs view PVM as a general and flexible parallel computing resource that supports a message passing model of computation. This resource may be accessed at three different levels, the *transparent* mode in which tasks are automatically executed on the most appropriate hosts, the *architecture-depend* mode in which the user may indicate specific architectures on which particular tasks are to be executed, and the *low-level* mode in which a particular host may be specified. Such layering permits flexibility while retaining the ability to exploit particular strengths of individual machines on the network.

While parallelizing an application on a multiuser network environment we have to deal with several problems not existing on a parallel computer. Here the effect is on both communication and computational performance of the program. As the machines are of different power, if we divide the problem into identical pieces one for each machine then the application will run as slow as the task on the slowest machine. If the tasks coordinate with each other, then even the fast machines will be slowed down waiting for the data from the slowest machine. The long message latency across the network also affects the performance of the application. As the performance of the network and the machines are dynamically changing the conditions are difficult to reproduce, and hence it is difficult to debug the application.

There are multiple ways by which we can distribute the tasks amongst different processors on the network.

In the simplest case if we have the number of processors equal to the number of tasks, we can assign them one each statically. Here the assignment may occur ofline even before the job is started. This kind of distribution is only useful when all the tasks have to be running together and also there is communication between them. In our application we do not have this kind of requirement and hence we will not consider this scheme any further. It also requires that the number of processors be equal to the number of tasks. This is not practical in our case as we generally have a very large population of tasks.

The other scheme, which we have implemented is based on the method known as *Pool of Tasks* paradigm. It is typically implemented in a master/slave implementation where the master programs creates and holds the "pool" and farms out tasks to slave programs as they fall idle. The pool is implemented as a queue and if the tasks vary in their sizes then the larger tasks are placed near the head of the queue. With this method all the slave processes are kept busy as long as there are tasks left in the pool. Since tasks start and stop at arbitrary times with this method, it is better suited to applications which require no communication amongst slave programs and the only communication that takes place is with the master or through data files.

Our requirement exactly matches with this model. Each generation in SPHC algorithm has a population of solutions to be evaluated, like a pool of tasks. We have a limited number of processors, so initially each processor is given a solution to be evaluated. As soon as any one of the processors finishes the task (as it happens frequently due to the difference in network load and computational power of machines) it is assigned another member of the population for evaluation. This way all the processors are kept busy.

The main SPHC algorithm when ready with all the solutions to be evaluated, makes a call to the master program. Simulation programs are kept on different machines which take part as slaves. These slave programs are spawned under the control of the master. PVM provides a library routine which allows the processes to be spawned on different machines on the network. The master then sends the appropriate data to this spawned task through message passing routines provided by PVM. We require to pass the structure representing the solution(controller). The slave program then runs the simulator with this controller, calculates the fitness of the solution and returns the fitness to the master. Master keeps one to one correspondence between the solution it had earlier passed to the slave and the fitness it returned. This is done by passing the solution number in the population also as a message to the slave. The slave then returns the same number. This way each fitness value produces its identification to the master process. This avoids the need for processes to finish in the order they were spawned. This is continued till there is no solution left in the pool to be evaluated. Once all the solutions are evaluated the master passes them with their fitness to the SPHC algorithm. The parallel form of SPHC algorithm is shown in Figure 6.11 and the master program which distributes the tasks is shown in Figure 6.12

We achieved considerable performance enhancement using this parallel SPHC algorithm. For example controller synthesis tasks which would take about 8 hours of elasped time on a single CPU took just 1 hour when the tasks were

```
Initialize population
Call Master to Evaluate in parallel all the solutions in the population
for generation = 1 to number of generations   do
    for each individual solution  in the population   do
        Randomly purturb the solution
    Call Master to Evaluate in parallel all the solutions in the population
    for each individual solution  in the population
        if  the new solution is better than the old one then
            Replace the old solution with the new one
        end if
    end for
    if (generation mod reseed-interval) = 0) then
        Rank order the population
        Replace bottom 50% of the population with top 50%
    end if
end for
```

Figure 6.11: Parallel stochastic population hill climbing (SPHC)

```
Determine the number of hosts currently available
Mark all the available hosts as free
while all the currently running tasks are not over OR there are tasks left to be evaluated do
    while there are tasks left to be evaluated do
        if there is a free host then
            spawn the next task on the host
            send the solution to be evaluated to the spawned task with its number in population
            mark host as busy
        end if
    end while

    if there are tasks running currently then
        wait for any of the task to get over
        get the host corresponding to this task
        mark host as free
        wait for slave message containing fitness value and solution number
        store the fitness value corresponding to the solution number
    end if
end while
```

Figure 6.12: Master program distributing the solutions on different hosts

farmed out to 8 workstations. The main feature of the master program is that it is independent of the number of hosts currently in the configuration. So we can add as many hosts as we want in the configuration and get better and better performance. The PVM system provides library calls by which we can identify the situation when a host is added into the configuration, or deleted from the configuration. We can use this feature to implement fully dynamic configurability.

6.3 Simulation of Motion and Sampling of Frames

Once the controller is obtained, it is plugged into the simulator to generate the motion and the sample frames at required rates. The default sampling rate is 25 frames per second of simulation. The frames are recorded in the format:

```
show <x> <y> <th1> <th2> <th3> ...
```

where	<x>	– is the x position of the root link.
	<y>	– is the y position of the root link.
	<th1>	– is the orientation of the root link.
	...	

Each individual frame is recorded as one command line and basically contains the values for each of the DoFs of the articulated figure.

6.4 Motion Playback

The frames computed and stored by the simulator are played back in real-time by "anix" a public domain animation server for X–windows [98]. This is an X-Windows program, and can produce real time output on a screen for display purposes or in a postscript file format for documentation purposes. The input to anix program is the output file created by simulator. At the end of display of each frame, the frame is erased by setting <erase> flag to true.

```
aniset erase t
```

6.5 Experimental Details

We describe below the structure and motion behaviour of three creatures that we have experimented with. These are named as Luxo, Pogo and Walker. Among these creatures, Luxo is the simplest creature made of just three links and two actuator joints. It is a one legged virtual creature. Appearance wise it is similar to a lamp and the only mode of locomotion available to it is hopping. Pogo a two legged virtual creature, is made of 5 links and 4 actuator joints. Appearance wise it is similar to a dog. Dynamically it is more stable than Luxo and can demonstrate different gaits such as walking and running. Walker is a human like virtual creature but without hands or head. It is made of 7 links and 6 actuator joints. In comparison to both Luxo and Pogo it is dynamically very unstable and this makes it very difficult to get a good controller which results into a steady walk.

The motion synthesis process is primarily based on the SPHC algorithm which takes time proportional to the number of generations G, the size of the population M, the amount of time to simulate for each trial, and the accuracy of the integration. This is because each individual controller's motion behaviour must be evaluated by simulation of the dynamics. Even if the simulation can be done faster than real time, it still must be performed for roughly $M \times G$ different controllers. In our computation, an 8 second controller simulation took approximately two minutes on VAX 8600. Though this may not be particularly fast in terms of CPU time, it is very efficient in human animator time.

For the documentation of the Animation System, refere to Appendix B. Details of the scripts describing for synthesizing

6.5.1 The Luxo creature

Figure 6.13 shows the geometric structure of Luxo.

Table 6.1 shows the allowable ranges of joint angles (in degrees) defining the internal configurations of Luxo.

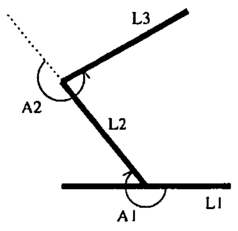


Figure 6.13: The articulated figure – Luxo (a lamp like creature)

Joint	*Min* (*deg.*)	*Max* (*deg.*)
A1	-300	-240
A2	360	210

Table 6.1: Range of angles for Luxo

Table 6.2 shows the physical properties of different links of Luxo.

Link	*Mass* (*kg*)	*Inertia* ($kg.m^2$)	$cmass_x$ (m)	$cmass_y$ (m)
L1	0.15	0.00312	0.0	0.0
L2	0.10	0.00208	0.25	0.0
L3	0.30	0.00625	0.25	0.0

Table 6.2: Physical properties of Luxo

Geometric Description Script for Luxo

```
# polyline definition of link object
objname baselink
path
 pts -0.27 0.02 0
 pts 0.27 0.02 0
 pts 0.27 -0.02 0
 pts -0.27 -0.02 0
objname middlelink
path
 pts -0.02 0.02
 pts 0.52 0.02
 pts 0.52 -0.02
 pts -0.02 -0.02
objname toplink
path
 pts -0.02 0.02
 pts 0.42 0.02
 pts 0.30 -0.15
 pts 0.47 -0.15
 pts 0.42 -0.02
 pts -0.02 -0.02
# instance of links
instance baselink link1
instance middlelink link2
instance toplink link3
# relative coordinate frame definition
attach link1 to world
attach link2 to link1
attach link3 to link2
# placement of each link
tf trans link2 0.0 0.0
tf trans link3 0.5 0.0
```

Physical Description Script for Luxo

```
# physical parameters of the link
link 1 0 0.0 0.0 0.15 0.003123 0 0
link 2 1 0.0 0.0 0.10 0.002082 0.25 0
link 3 2 1.0 0.0 0.30 0.006246 0.25 0
# specification of monitor points
monitor 1 1 -0.27
monitor 1 2 0.27
# file name containing equations of motion code
set procname luxo
# compiling equations
compile
quit
```

Simulation Script for Luxo

```
# simulation time step
set dtsim 0.001
# symbolic code for equations of motion
dyn luxo
# symbolic code for monitor points
mon mon_luxo 2
# number of state variables
set state_size 10
# initial values of state variables
state 0.0,0.0,0.0,0.0,0.0,0.0,-4.9,0.0,4.665,0.0
# name of the output file
set dispfile luxo.out
# simulation time (sec)
sim 10.0
quit
```

Animation Script for Luxo

```
# initialize server
init
# read the values x  y  th  th1  th2  for a frame
tf trans link1 _1 _2 0
tf rot link1 Z _3
tf rot link2 Z _4
tf rot link3 Z _5
# initial position of link1 on the screen
tf trans world 10 4
# scale the articulated figure
tf scale world 3 3
# set degree mode to false
aniset degmode f
# erase the frame at the end of the display
aniset erase t
```

The controller for Luxo has been designed using a two node pose control graph. The parameter space is ten dimensional. The ratio k_p/k_v is chosen as 0.1.

$$[\theta_1^1, \theta_2^1, k_{p1}^1, k_{p2}^1, t_1] \qquad\qquad [\theta_1^2, \theta_2^2, k_{p1}^2, k_{p2}^2, t_2]$$

Five posture features to synthesize hopping motion are listed in Table 6.3. The postures are approximately 0.2 sec apart in time. The value of distance to be travelled in a single hop is given as $0.4(m)$ and external energy as $2.3(Nm)$. Just in order to convince ourselves that the SPHC algorithm will indeed find the optimum, we synthesized the same motion i.e optimizing the same performance metric by randomly choosing many different sets of initial populations. Figure 6.14 shows two such cases of the progress of the SPHC search algorithm in finding the hopping motion controller for Luxo. In the first case the desired controller is found after 40 generations with a population size of 50. In the second case, more or less a similar controller was found after only 25 generations with the same population size.

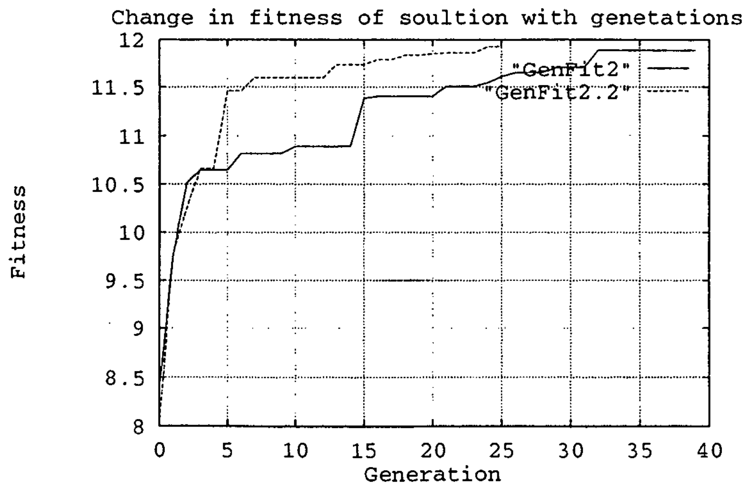


Figure 6.14: Synthesis of two different controllers for Luxo

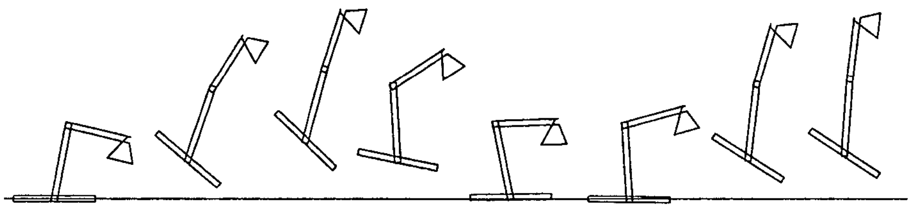


Figure 6.15: Luxo hopping

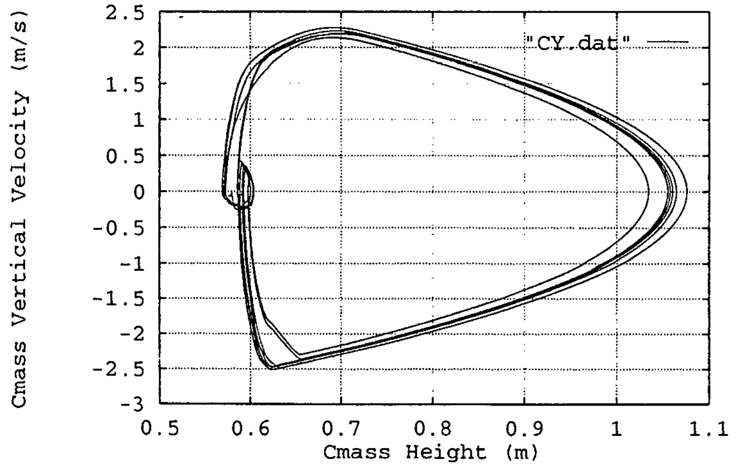


Figure 6.16: Phase diagram for hopping Luxo

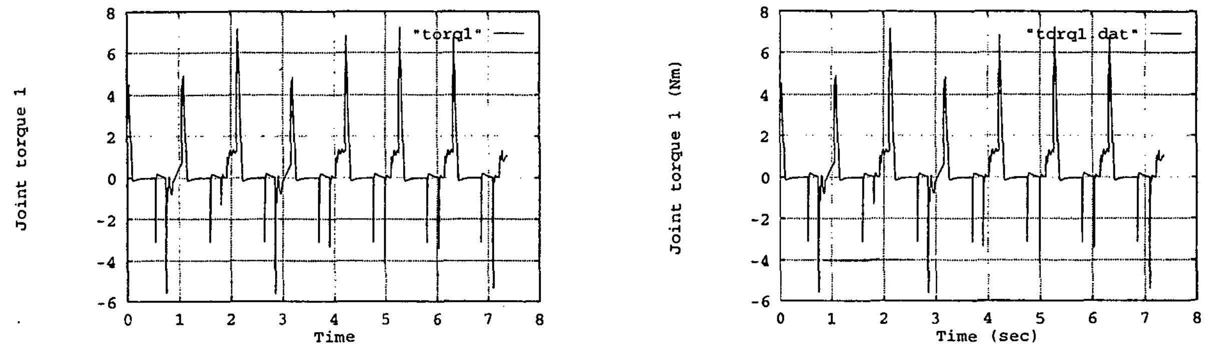


Figure 6.17: Variation in torques for hopping Luxo

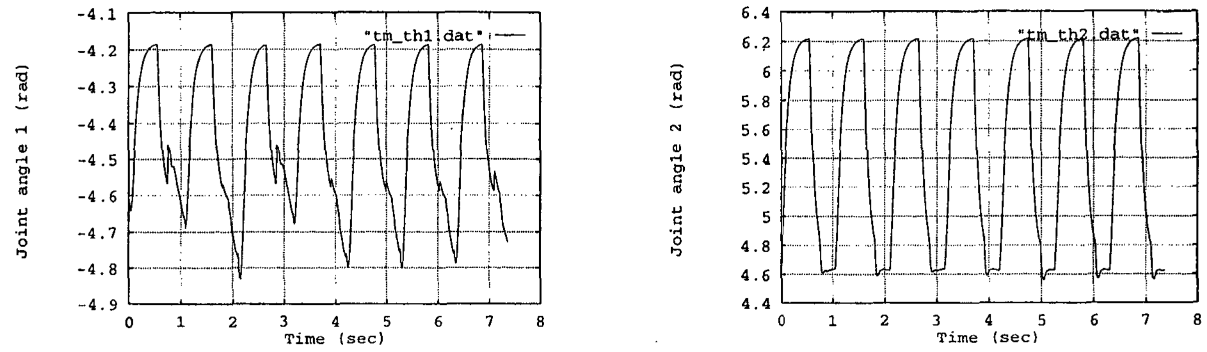


Figure 6.18: Variation in joint angles for hopping Luxo

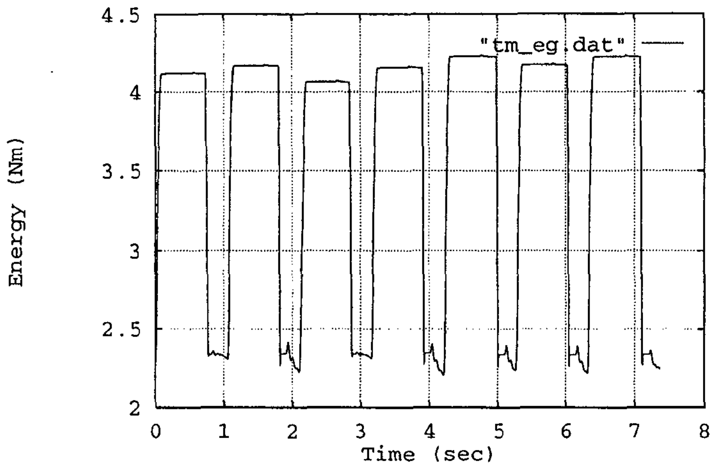


Figure 6.19: Variation in total energy of hopping Luxo

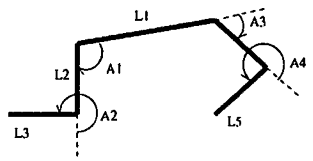


Figure 6.20: The articulated body – Pogo (a dog like creature)

Joint	*Min* (*deg.*)	*Max* (*deg.*)
A1	-115	-45
A2	270	360
A3	-70	-40
A4	210	300

Table 6.4: Range of angles for Pogo

values have been chosen so as to synthesize walking as well as running motion. The progress of the search algorithm in finding the walking motion controller is shown in Figure 6.21.

Two different gaits are synthesized for Pogo. In the first experiment a walking gait is obtained by choosing four intermediate postures at time intervals of approximately 0.2 sec as shown in the table 6.6. The value of distance travelled and external energy in a single gait cycle is chosen as $0.6(m)$ and $0.7(Nm)$ respectively. The output is shown in the Figure 6.22 below.

Figure 6.23 shows a phase diagram which plots height of centre of mass versus vertical velocity of centre of mass. The trajectories in the phase diagram once again show a periodic behaviour with trajectories being attracted towards an attractor cycle, indicating stable behaviour. Figure 6.24 shows the variation in total energy for walking Pogo.

In the second experiment a running gait is obtained by choosing three intermediate postures (Table 6.7) with value of distance travelled in a single gait cycle as $0.5(m)$ and external energy as $0.8(Nm)$.

Link	*Mass* (kg)	*Inertia* $(kg.m^2)$	$cmass_x$ (m)	$cmass_y$ (m)
L1	0.15	0.003123	0.25	0.0
L2	0.10	0.002082	0.125	0.0
L3	0.10	0.002082	0.125	0.0
L4	0.10	0.002082	0.125	0.0
L5	0.10	0.002082	0.125	0.0

Table 6.5: Physical properties of Pogo

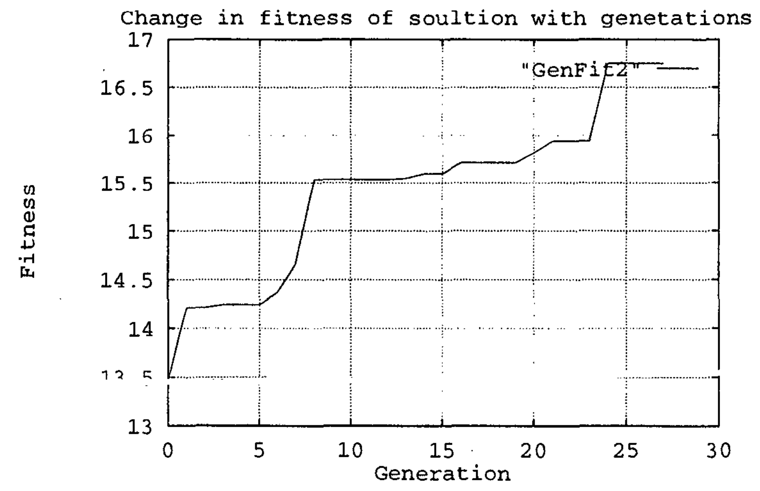


Figure 6.21: Synthesis of a controller for Pogo

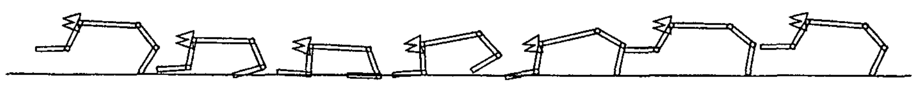


Figure 6.22: Pogo, walking

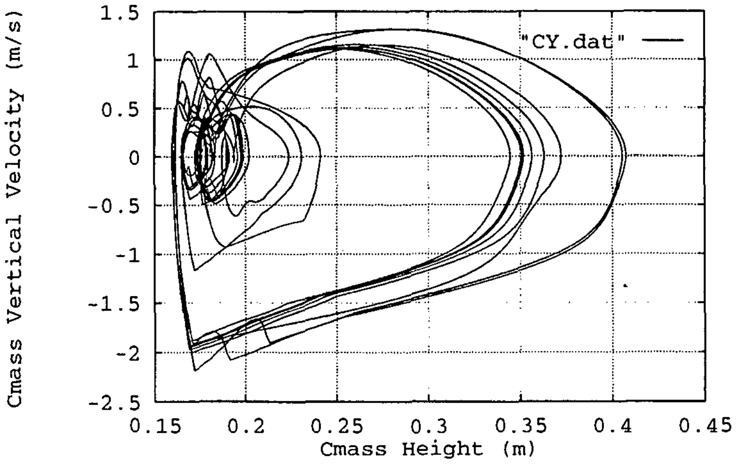


Figure 6.23: Phase diagram for a walking Pogo

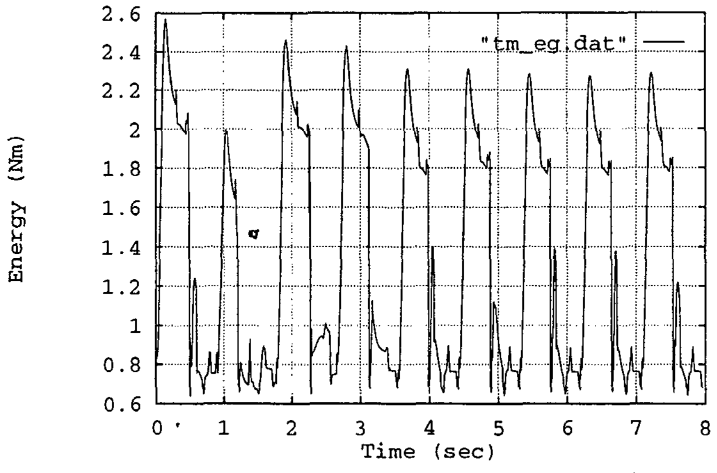


Figure 6.24: Variation in energy for walking Pogo

$Pose\#$	θ_1 $(rad.)$	θ_2 $(rad.)$	θ_3 $(rad.)$	θ_4 $(rad.)$
1	-1.9700	5.1936	-0.8497	5.0566
2	-2.0709	5.2423	-0.6785	5.2343
3	-1.8970	4.7794	-0.8469	4.7110
4	-1.5962	4.7227	-1.1903	4.3722

Table 6.6: Four posture features for synthesizing a walking motion for Pogo

$Pose\#$	θ_1 $(rad.)$	θ_2 $(rad.)$	θ_3 $(rad.)$	θ_4 $(rad.)$
1	-1.0138	5.8728	-1.0830	4.1118
2	-1.8247	5.4664	-0.9404	4.6641
3	-1.9589	5.3294	-0.8445	4.7969

Table 6.7: Three posture features for synthesizing a running motion for Pogo

The output is shown in the Figure 6.25 below.

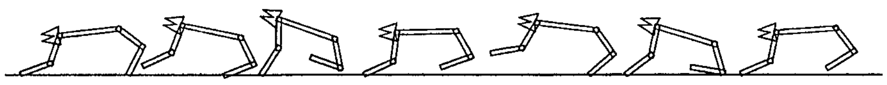


Figure 6.25: Pogo, running

Figure 6.26 shows a phase diagram for running Pogo. After initial instable behaviour, the trajectory settles in with a periodic behaviour. Figure 6.27 shows the variation in total energy for running Pogo.

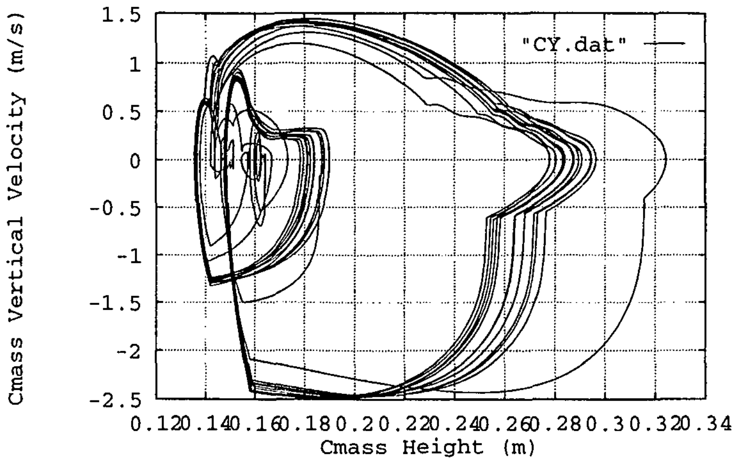


Figure 6.26: Phase diagram for running Pogo

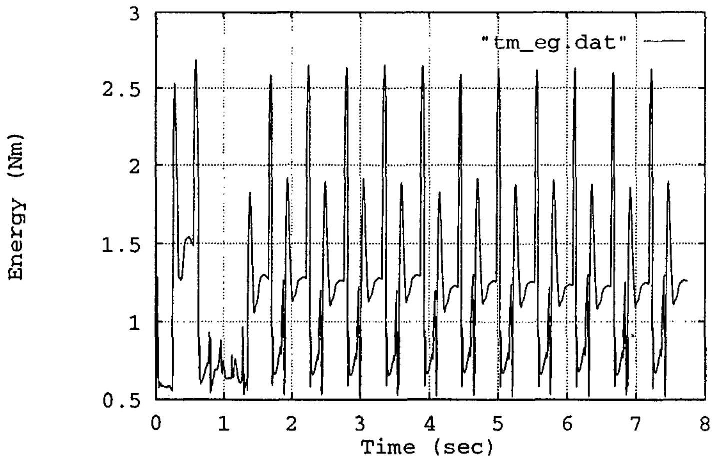


Figure 6.27: Variation in total energy for running Pogo

6.5.3 The Walker creature

Figure 6.28 shows the geometric structure of Walker. Controller for Walker consists of a four node pose control graph.

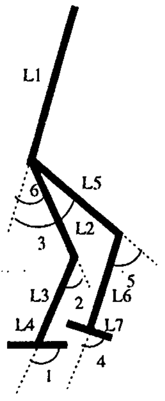


Figure 6.28: The articulated body – Walker (a human like creature)

Table 6.8 shows the allowable ranges of joint angles (in degrees) defining the internal configurations.

Joint	*Min* (*deg.*)	*Max* (*deg.*)
A1	-160	-125
A2	-10	-45
A3	60	100
A4	-160	-125
A5	-10	-45
A6	60	100

Table 6.8: Range of angles for Walker

Table 6.9 shows the physical properties of different links.

The result of the simulation of Walker is shown in the Figure 6.29 below.

Link	*Mass* (kg)	*Inertia* $(kg.m^2)$	$cmass_x$ (m)	$cmass_y$ (m)
L1	3.0	0.0625	0.250	0.0
L2	5.0	0.0260	0.125	0.0
L3	4.0	0.0208	0.125	0.0
L4	5.0	0.0260	0.0	0.0
L5	4.0	0.0208	0.125	0.0
L6	1.0	0.0052	0.125	0.0
L7	1.0	0.0052	0.0	0.0

Table 6.9: Physical properties of Walker

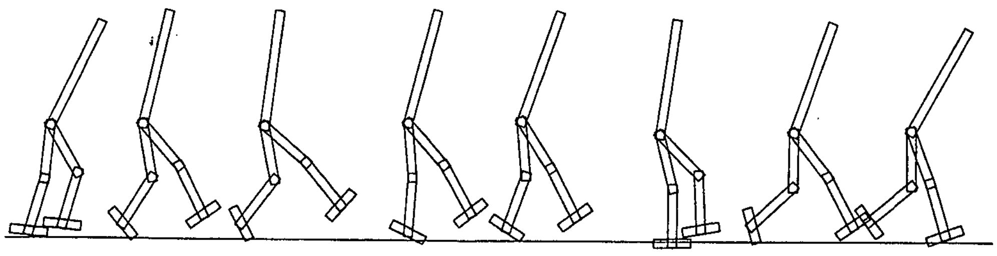


Figure 6.29: Walker walking

Chapter 7

Conclusions and Future Work

This research has been concerned primarily with the field of computer animation. Traditionally computer animation has been a tedious and time consuming process. The animator is given minimal assistance by the computer animation systems that are in wide use even today. Physically based animation, particularly automatic motion synthesis – where by the animator specifies only the physical and geometrical structure of the character and the criteria for evaluating the character's motion in the environment, and the computer generates a physically correct, realistic and natural looking trajectory for the character–is very much more attractive, though computationally very difficult to realize and operationally very difficult to control.

As part of this research we have carried out a detailed investigation of prior work in automatic motion synthesis. The most promising approach to the motion synthesis problem has been identified as the automatic generation of a motion controller which when executed in a simulated physical environment produces a desired motion. We have analyzed different representations for motion controllers, the different optimization techniques and also the techniques for simulation of movement along with interaction with other environment objects. We have identified the primary reasons why animators do not get fine control over the synthesized motion. As part of the results of our research we have invented a novel method of providing fine control over the synthesized motion while at the same time retaining all the advantages of automatic synthesis of physically realistic motion. In this last chapter we first discuss the significant contributions that have been made as a result of this thesis research and also the limitations of our present method, and its implementation. We end this thesis with a discussion on possible future extensions of our research and some open problems in this domain.

7.1 Significant Contributions

The most significant contribution in this thesis is certainly the new formulation of the motion specification problem as one of motion feature specification. The method of specification of desired motion, through the input of features characteristic of the desired motion, is novel, very elegant and we believe the most convenient method of providing complete control to the animator in his/her use of the computer for creating complex animated character sequences. By first specifying only high level features and then gradually adding greater detail level features the animator gets progressively finer control over the generated animation – a highly desirable feature in any interactive design.

The observed behaviour of various physical phenomena around us has always been a major source of inspiration for humans whenever we have had to understand, explain or mechanically imitate these natural phenomena. By asking the animator to synthesize motion by specification of high level motion features as observed by him/her in the different types of movements that are taking place in the real world around us, this method is inherently intuitive and can therefore provide very natural interfaces to the motion synthesis task.

We have formalized the notion of motion features as computable functions, and formulated a performance metric for any motion such that the metric attains an optimal value whenever the desired features are best present in that motion. By implementing the above formulation and experimenting with the features present in different gaits of legged creatures, we have convincingly demonstrated that given adequate desirable features, motion can be automatically synthesized such that the synthesized motion has all the given features, is physically realisable and also realistic. This now reduces the burden on the animator tremendously. The animator has only to interactively tune the feature values or seek additional features in the synthesized motion until it is completely to his/her satisfaction.

Another significant contribution of this research has been the creation of an integrated simulation environment that has the capability to synthesize motion controllers from feature specifications and to generate the final sequence for playback of the animation. Building such an environment has in itself required enormous efforts. While the environment does not have at present any fancy user interface, it has all the core components necessary for synthesizing the motion of articulated figures. This includes figure representation, controller representation and execution, force/torque synthesis using the spring/damper model for muscle actuator behaviour, automatic symbolic formulation of the dynamic equations of motion, solving the dynamic equations of motion and integrating acceleration/velocities to obtain the individual image frames that make up the animation. These core components are sufficiently general and will be extremely useful to experiment with other methods of motion synthesis as well.

The third significant contribution is in the design and implementation of a parallel evolutionary programming algorithm, the stochastic population hill climbing algorithm used in the global optimization search for the motion controller with the desired motion features. The parallel SPHC algorithm has been implemented in a distributed network of heterogeneous workstations using the parallel virtual machine PVM system. The parallel SPHC has resulted in very substantial reduction in computational time and has enabled experimentation with reasonably complex virtual creature moving in a variety of gaits.

In addition to the above, the thesis has a very comprehensive survey of all known methods for physically based animation, and motion synthesis. These methods have been presented in the formal frame-work of non-linear optimized search. This frame-work enables us to analyze all existing techniques and also newer motion synthesis techniques that may be developed in future.

7.2 Limitations

The automatic synthesis of physically based motion is a multidisciplinary field and has required the understanding and implementation of formalisms and techniques from very diverse fields including computer graphics, computer animation, computer vision, mechanical simulation, robotics, neural networks, computational neuroethology, non-linear optimization and biomechanics. Given constraints of time and other resources there are bound to be many limitations. Below we discuss some of the limitations in our approach, in our implementation and in our experiments.

A major limitation in the basic approach which is also present in all other motion synthesis methods in use today is the specificity of the synthesised motion. While we do design a motion controller, the motion controller's behaviour is situation optimized. So if we are able to synthesize a motion controller, say, for walking on a floor inclined at an angle of 5 degrees to the horizontal, then, in general, the same controller cannot be reused for walking on floors with other inclinations. It has been argued in [102] that by suitably parameterizing the motion controllers it should be possible to reuse the synthesized motion controllers in similar conditions with only slight differences. We however believe that motion controller specificity is an inherent limitation of the approach and can be overcome only if generic movement behaviours can be learnt. The virtual creature has to learn the primitive behavioral mechanisms needed for walking. This, as we shall see later, is an open problem.

With respect to the implementation we have some deficiencies in the simulation environment as well as the synthesis method. The simulation environment supports only 2D articulated figures. Further the articulated figures have to be tree structured. Joint actuators are modelled using the spring/damper mechanism but not using the biomechanics based muscle model. Joint types have all to be pin-joints. Many of these deficiencies are not very severe and have been introduced primarily to keep the entire implementation simple and computationally tractable within the available computing facilities. The simulation environment can certainly be augmented to remove these deficiencies. While it may prop up some difficulties and may involve considerably extra efforts, there do not seem to be any unsurmountable problems.

There are some limitations in the method we have implemented for automatically synthesizing motion controllers. One of these is the static nature of the controller representation during the entire process of searching for an optimal motion controller. We assume for example in the case of a pose control graph type motion controller that the topology of the controller (number of nodes and their connectivity amongst them) is fixed even before the start of the search process and remains invariant there after. Clearly the domain representable motions of a pose control graph controller with 3 nodes is larger and differs from that of a controller with 2 nodes. This automatically introduces artifacts in the final synthesized motion which are inherently not present in the approach. Instead of using an n node pose graph controller for synthesizing some desirable motion, an m (where $m \neq n$) node pose graph controller may provide much better results. The search process could be modified to include the controller topology also as part of the parameter space to be searched. This would however give rise to a considerable increase in the search space dimensionality and result in reduced efficiency.

A more serious limitation of our synthesis method is the fact that the motion is synthesized for a single virtual creature moving in a static unchanging environment. For example, our method, as currently implemented, cannot be used to synthesize the movement of a creature catching a ball in flight. This specific case may not be very difficult to incorporate. It could be done by suitably formulating the dynamic equations of motion for the ball as well, and including them in the system being simulated. The performance metric will also have to be suitably modified to include minimization of distance between the creature and the ball and also an end condition in which this distance is zero. In general however, it is difficult to visualize how the method can be extended to consider a changing environment consisting not only of passive objects but also active creatures (other virtual creatures) carrying out their own movement. As we shall soon see this is yet another open problem in this field.

While the experiments that we have conducted to synthesize the gaits of a number of different virtual creatures have led us to believe that this method of motion synthesis is well suited to character animation, we do know that many more experiments are needed before the method can be put to practice on a regular basis. We have been able to study the gait related features in some detail and use some of them to automatically synthesize walking/running/hopping/falling types of motion. We have not however experimented using all of them for motion synthesis and determined their effectiveness/criticality towards the synthesis task. For example at this point of time we are unable to associate any discrimination quality characteristics with any of the features. Clearly other experiments have shown the comprehensive discriminating capabilities of features like the Froude number or the duty cycle. As we collect more experimental statistics working with features, it should be possible for us also to make our observations.

We have been fortunate that the study of gaits of legged creatures is a subject that has been addressed rigorously by a large number of researchers in different disciplines. As a result we could draw upon their research and arrive at a comprehensive set of features that could be used to specify different kinds of movements of legged creatures. There are other goal oriented movements like catching a ball, hitting a ball, shotting into a goal post, dunking a basket, picking an object etc. All these would require their set of features to be suitably identified/desired.

7.3 Possible Extensions

We have seen that many of the deficiencies and limitations described above are in themselves pointers for interesting extensions to our research, to our motion synthesis method and also to the implementation of the simulation environment. There are some other important extensions which will definitely need to be addressed in the near future if the method has to be used more widely, These we briefly discuss below.

In general, the representation of the controller and the synthesis method are such that the actuator response is programmed for a specific external environment configuration. It would indeed be very interesting to explore a generalization of the controller representation schemes so that external objects and interaction events are embedded into the motion controller along with actuators so that the actuator response could be made more general and less situation specific. For example consider the case of our virtual creature, Walker moving on a floor which could be inclined within a range say, -15 to 15 degrees. The controller representation needs to be suitably generalized to take into account this slope in the floor and the synthesis method should be able to give us a controller with that representation which can produce similar looking movements on sloping floors. Clearly the motion controller representation has to be rich enough so that issues such as maintaining balance, not toppling over, etc. would have to get incorporated into the motion controller. Control representation for maintenance of balance has been a subject of detailed study by many others and it should be possible to draw upon some of those results [50, 87].

While the work in this thesis has dealt exclusively with the control synthesis problem for which stick figures are adequate, there is a need to dress up these virtual creatures with skin or clothes to produce interesting images. It needs to be determined how such skin should be attached to the mechanical skeleton and how it can be made to deform upon bending of joints or contact with other objects. Flexible skin could be surrounded or be controlled by the rigid components. Various materials could be added such as hair, fur, or tentacles that might flow or bounce, producing secondary motion effects which will add to the overall realism of the motion.

In addition to the extensions to the implementation discussed in the earlier section the following would also be needed

- A good user interface which will enable the animator to specify features, constraints and other simulation parameters graphically and also provide the user with good analysis and visualization tools for evaluating the synthesised motion. The system should be able to store simulation results and play back after concatenating them, if so desired by the animator.

- A good technique for compositing motion sequences generated through automatically synthesized controllers also needs to be incorporated into the implementations. Different topological arrangements of controllers, linear, hierarchic, network etc. will have to be supported and appropriate mechanisms for execution control transfer will also have to be built in. This will be needed if complex animation sequences have to be built out of individually synthesized motion controllers.

- If the skeletal structures are fleshed out with skin/cloths then three dimensional rendering and visualization facilities with control over optical properties of surfaces, the illumination in the environment and texture mapping will have to be incorporated for realistically rendered animation sequences.

- In our implementation we have paid considerably more attention to the modelling of motion controllers and the synthesis of internal forces and torques. The external environment and interaction modelling are equally important in motion simulation. Thus collision detection algorithms have to be made more general and multi-point collisions have to be handled. More sophisticated collision response behaviour modelling has to be incorporated, including the difficult problem of handling of frictional contacts.

7.4 Open Problems

Our research has thrown up a number of open problems. Primary among these are the learning of motion behaviour, synthesis of multiple virtual actor motion and the existence of a basis set for motion features. We discuss each of these in a little more detail below.

Learning Motion Behaviours

As we have pointed out earlier, when the motion controller is automatically synthesized it basically embeds into itself a situation specific behaviour. Thus by giving the appropriate feature values to the motion controller associated with our virtual Walker, it can be made to walk on the floor. However in no way can it be said that Walker has learnt to walk. Learning motion behaviour is certainly an open problem. For this we believe that it would be essential to consider synthesizing complex movements through the logical composition of a few primitive motion behaviour controls. What must these primitive controls be or what is the logical composition mechanism necessary are certainly unsolved problems.

The Motion Synthesis of Interacting Multiple Virtual Actors

The complexity of the motion synthesis problem increases in an unbounded fashion when one considers interaction not only between a virtual creature and a static or dynamic (but passive) environment but also between two or more virtual creatures. There are two situations which need to be dealt with.

- a common goal situation in which all the virtual creatures have the same goal. Examples include two virtual creatures moving towards each other to meet, or a team of players particularly in a football game, or a troupe of ballet dancers performing in perfect synchrony.
- a conflicting goal situation in which groups of virtual creatures are at cross purposes. Examples include one virtual creature being chased by another and trying to avoid being caught, or a game of doubles tennis.

The complexity underlying these behaviours is just unfathomable. Identifying the important features or formulating a suitable performance metric for the above types of behaviours is certainly an open problem.

The Universal Set of Features

Finally there is also this open question of whether there exists a finite universal set of features which form a basis for describing all movements of a particular kind, say, gaits of four legged creatures. The existence of such a feature basis would certainly imply completeness in motion synthesis by the method of specifying features. Until then we must assume that certain movements will always elude formalism.

Appendix A

Representation, Performance metric and Mutation Operation

This appendix describes in detail the representation of an individual solution used in the evolutionary algorithm. It also gives an intuitive explanation of various terms used in the performance metric and the reasons and motivation for choosing them. Further, the mutation operation which is used to manipulate the individual solution is also documented in detail.

A.1 Representation of Solution

In evolutionary programming, the representation of individual solution plays a very crucial role in the over all performance of the algorithm. In our method, an individual solution represents a controller. We have chosen a Pose control graph to represent a controller. Pose control graphs have been described earlier in section 2.6.2.

The arcs of the pose control graph specify the fixed time interval upon which the transition between the states takes place. The desired pose associated with the state is kept fixed for the duration of the time interval.

A pose control graph can be represented using a set of parameters called solution vector. All these parameters have direct influence over the motion produced by the controller, and so have to be chosen carefully for a particular motion. The aim of the evolutionary algorithm is to synthesize these values for a required motion. We have used the following parameters in our representation of pose control graphs.

- Poses represent the internal configuration of the body. Internal configuration is nothing but a set of joint angles which can be varied over time. In the case of a n link tree structured body there can be maximum $n-1$ joints. Each pose is defined as some combination of these joint angles which are to be achieved after a specified time interval. If there are m poses defining the pose control graph, there will be m sets of $n-1$ joint angles. So the solution vector representing the controller will have the combination of $m(n-1)$ joint angles.

- Spring and damping constants. The torque applied at a joint is given by the function
$$\tau = k_p(\theta_d - \theta) - k_d\dot{\theta}$$
It is proportional to the difference between the desired joint angle and the current angle. Also spring and damping constants k_p and k_d can significantly affect the type of motion. Hence we have also included some of these constants in our solution vector [1]

- Time interval between poses. This parameter plays an important role. We want the creature to achieve the desired pose but before it achieves the desired pose the time interval will get over and the state will change. Once the state changes the current configuration will be compared to the parameters associated with that state.

A three link Luxo creature with two joints and two pose control graph is shown in the figure A.1. The solution vector for such a representation is

$$[\, t_1\ \theta_d^{11}\ \theta_d^{12}\ k_p^{11}\ k_p^{12}] \quad [\, t_2\ \theta_d^{21}\ \theta_d^{22}\ k_p^{21}\ k_p^{22}]$$

[1]Experimentally a value of damping constant as one tenth of spring constant has been found to be suitable.

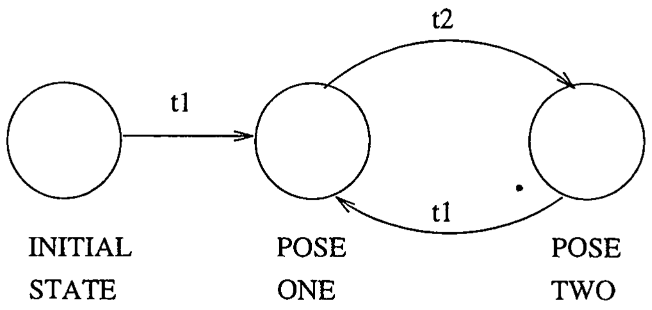


Figure A.1: Transition between Poses

Given an initial state a transition is made to pose1 ($\theta_d^{11}, \theta_d^{12}$) in time interval $t1$, with values of k_p^{11} and k_p^{12}, then a transition from pose1 to pose2 ($\theta_d^{21}, \theta_d^{22}$) in time interval $t2$, with values of k_p^{21} and k_p^{22}. After that the cycle repeats. Every time pose change it cause certain torques to be generated which form the input to the simulator.

An ideal automated synthesis system would be able to design a controller given only the mechanical structure of the creature, and no other *a priori* information. However, by specifying a small but useful amount of additional information it becomes possible to greatly reduce the search time and improve the performance of the algorithm. The additional information we are providing is as follows:

1. the ranges of time interval between the poses
2. the number of poses required for the motion, and
3. the expected ranges of spring constants

These numbers are not very difficult to estimate. An estimation of the time interval is done based upon the size and shape of the creature. Values of spring constants are estimated with the help of mass description of the body. If the links between which the spring has been simulated are having higher weight, higher valued ranges of spring constants are required in order to protect the springs from a possible collapse(spring failure). Also values should not be so high that they would cause high torques at the joint causing unexpected motions.

Estimating the number of poses for a creature is comparatively more difficult task and usually needs in depth knowledge of expected motion of the creature. Experience also helps in deciding this number.

Given the above mentioned information, the synthesis technique must find the controller that will perform well with respect to a given performance metric which in turn leads to a desired motion.

A.2 Performance metric

In a broad sense, the synthesis process searches through a space of controllers and selects the best one satisfying the motion features specified by the animator. How good the match is, is determined by the performance metric. The performance metric typically evaluates the controller by plugging it into the simulator and generating the motion. We have considered following performance metric:

$$f = w_1 * \sum_{s=1}^{samples} \sum_{j=1}^{joints} (1 - (\theta^o_{(s,j)} - \theta_{(s,j)})) + w_2 * (1 - \frac{(E_o - E)}{E_{max}}) + w_3 * (1 - \frac{(D_o - D)}{D_{max}})$$

where,

w_1, w_2, w_3	are weights, assigned to the features depending on their relative importance, the value ranging between 0 and 1.
$\theta_{(s,j)}$	is the angle at joint j for posture,
E	is the external energy
D	is the horizontal distance travelled
E_{max}, D_{max}	are the maximum expected external energy and horizontal distance, respectively. These values are used for normalization of the two quantities.
$\theta^o_{(s,j)}, E_o, D_o$	are the feature values specified by the animator.

It is clear from the above function that, closer the match between the specified and synthesized feature values, higher would be the value of the performance metric. The specified and achieved poses are compared during each cycle and the difference is summed up. The external energy and the horizontal distance are summed during each cycle.

Poses provide the actual snapshot of the object at an instance, like keyframes. In order to get similar kind of motion, these values have to match as close as possible with the corresponding controllers values. Thus, higher weight w_1 is given to this comparison. External energy and horizontal distance traveled help in optimization process in case of ties in other feature values. For example, object can achieve the same configuration without even moving from its place in the environment. Generation of such a motion has been largely averted by using external energy and horizontal distance features.

A.3 Mutation Operation

Mutation operation is the backbone of SPHC algorithm. In every iteration all solutions go through this operation. Each solution is represented by a set of parameters as discussed above. These parameters are time interval, desired joint angles and spring constant parameters.

Change in one of the parameters can change the motion drastically. So a careful mutation of these parameters is required with only a small change in original value. We have selected to mutate only one parameter in the whole state once the mutation operator is applied. The selection of this parameter is a randomized process, with equal probability is given to all the parameters. This is found suitable through experiments as it helps the algorithm to fully explore the region near an existing solution. If we try to mutate more than one parameters at a time, the solution may jump from one region to another without exploring the current one. As the function is multimodel, it may be the case that optimal solution is in the vicinity of solution being mutated.

Each time mutation is called the selected state goes through a *creep* operation. In creep operation a randomly selected parameter is modified with a very small factor or all its parameters are randomized from scratch. As there are three types of parameters to be modified, there are three possible creep operations which are defined as follows:

1. The original time interval is multiplied by a randomly chosen factor close to unity (0.8 – 1.2).
2. One of the joint angles is selected randomly and changed by a randomly chosen amount between -10^0 and 10^0, and
3. One of the joint angles is selected randomly and multiplied by a randomly chosen factor close to unity (0.8 – 1.2).

Appendix B

Animation System Reference Manual

This appendix list the syntax of all the commands supported by different modules constituting the experimental animation system used for synthesizing the animations in this thesis.

B.1 Symbolic equation generator

```
Command summary:
< <filename>
        Reads the script in the given file as input.
cat <filename>
        Prints out the given file
compile
        Compiles a procedure for solving the equations of motion.
echo <args>
        Prints arguments
fix <link_num> <fix_x> <fix_y>
        Fixes the location fix_x, fix_y on the given link.
link <link_num> <parent_num> <attach_x> <attach_y> <mass> <inertia>
 <mass_x> <mass_y>
        Creates link number link_num, attached to the given parent at
        the given point, having the given mass, inertia
        (with respect to centre of mass) and centre of mass.
monitor <link_num> <mon_num> <x> <y>
        Allows for external forces to applied at the given x,y on the
        given link.
quit
        Exits the program shell
set <var> <value>
        Sets the variable to the given value.
        The values are obtainable by executing 'set'
        without any arguments
procname   STRING
        name of desired procedure
```

B.2 Simulator

```
Command summary:
< <filename>
        Reads the script in the given file as input.
cat <filename>
        Prints out the given file
dyn [proc]
        Chooses the given dynamics procedure to be used.
        With no arguments, it lists the current dynamics
        procedures available.
echo <args>
        Prints arguments
quit
        Exits the program shell
set <var> <value>
        Sets the variable to the given value.
        The values are obtainable by executing 'set'
        without any arguments
debug      bool    turns debugging info on or off
dispfile   string  name of file for output of display information
dtdisp     float   display time step
dtsim      float   simulation time step
kdamp      float   damping constant
state_size int     the size of the state vector
sim <time>
```

```
        Simulates the system for the given amount of time,
        in seconds. The simulation time step is given by
        'dtsim'. The display time step is given by 'dtdisp'
        (see the 'set' command)
state <x,vx,y,vy,th,dth,th1,dth1,th2,dth2...>
        Sets the current system state to the given value.
```

B.3 Anix server

```
summary of commands:
< <file_name>
Read input from file.

aniset <var1> <value1> <var2> <value2> ...
        Sets environment values to desired values
        Aniset without any arguments returns the current value
        of all the variables. A brief description of the use of
        all the accessable variables is as follows:
```

debug	flag to output excess information for debugging
degmode	flag to specify rotation transformations in degrees
device	"ps" for postscript, "display" for X-11 window
display	name of host to use as an X-11 server
erase	erase display between frames? (X-11 only)
eyedist	distance of eye from screen for perspective projection
helpfile	file containing documentation
newsfile	file containing list of recent modifications
psfile	name of file to send postscript output to
showtime	echo current time
sleep	time to pause between displaying frames
viewdist	distance of eye from viewpoint for perspective proj.
viewto	the viewpoint, placed in the centre of the screen
viewfrom	specifies line of sight
viewup1	which direction to consider 'up' for display purposes
viewup2	a second choice for an 'up' vector in case viewup1 is very close to being parallel to the line of sight
winx,winy	size of X-window
xwin,ywin	window placement

```
attach <objname> to <objname>
```

```
        Attaches one object instance to another object instance.
clear
        Clears the x-display
close
        Closes animation display.
comments
        All text following a '#' character on a line is ignored
detach <objname>
        Detaches the object from its parent
help [topic]
        Lists information in help file on the specified topic.
        Type 'help sum' for a summary of available commands.
init
        Calculate world to screen transformation, prepare for display.
instance
        Creates an instance of an object
news
        Print the most recent updates.
objectname <objectname>
        Begin a new object
path
        Begin a new path.
pts <x> <y> <z> ...
        Adds points to the current path.
quit
        Exit program.
show <val1> <val2> ... <valn>
        Supplies a series of missing transformation values and then
        displays using the new transformations.
sleep <time>
        Causes a pause for the specified number of seconds.
text x y z string
        Prints text at the given point
tf <obj> <tf_name> <tf_args>
        Performs the specified transformation on the object.
        The acceptable transformations (as specified by
        tf_name and tf_args) are:
rot     <x|y|z> <angle>
        Rotates the object about the given axis by the angle
```

in degrees.

prot <x|y|z> <angle>

Rotates the object about the given axis in the parent's coordinate system. The angle is in degrees.

trans <x> <y> <z>

Translates (moves) the object as specified in the objects own coordinate system.

ptrans <x> <y> <z>

Translates (moves) the object as specified in the parent's coordinate system.

scale <x> <y> <z>

Postmultiply the ctm by the given scaling factors. The object it self and all following transformations are affected by the scaling.

pscale <x> <y> <z>

Premultiply the ctm by the given scaling factors. All transformations and the object itself are affected by the scaling.

tpipe <file1> <pipefile>

Repeatedly pipes in <pipefile> while <file1> exits.

Appendix C

Documentation of Scripts

This appendix documents all the scripts for Luxo, Pogo and Walker in generating the animations. Scripts with extension .anix gives the geometric description of the creature. Script with extension .desc describe the physical structure of the creature and the script with extension .sim is used in producing the simulation and recording of the frames.

C.1 Scripts for Luxo

luxo.anix

```
aniset winx 600
aniset winy 600
objname baselink
path
        pts -0.27 0.02 0
        pts 0.27 0.02 0
        pts 0.27 -0.02 0
        pts -0.27 -0.02 0
objname middlelink
path
        pts -0.02 0.02 0
        pts 0.52 0.02 0
        pts 0.52 -0.02 0
        pts -0.02 -0.02 0
```

```
objname toplink
path
        pts -0.02 0.02 0
        pts 0.42 0.02 0
        pts 0.35 0.02 0
        pts 0.40 0.02 0
        pts 0.30 -0.15 0
        pts 0.47 -0.15 0
        pts 0.42 -0.02 0
        pts -0.02 -0.02 0
instance baselink link1
instance middlelink link2
instance toplink link3
attach link1 to world
attach link2 to link1
attach link3 to link2
tf trans link2 0.0 0.0 0
tf trans link3 0.5 0.0 0
< gnd.anix
init
tf trans link1 -3.0 0 0
tf trans link1 _1 _2  0
tf rot link1 Z _3
tf rot link2 Z _4
tf rot link3 Z _5
tf trans world 15 4 0
tf scale world 3 3 3
aniset degmode f
aniset erase t
```

luxo.desc

```
link 1 0 0.0 0.0 0.15 0.003123 0 0
link 2 1 0.0 0.0 0.10 0.002082 0.25 0
link 3 2 1.0 0.0 0.30 0.006246 0.25 0
monitor 1 1 -0.27 0
monitor 1 2 0.27 0
set procname fall
compile
quit
```

luxo.sim

```
set kdamp 0.0,0.0,0.0,0.0,0.0,0.0,0.0
set dtsim 0.001
dyn fall
mon mon_fall 2
set state_size 10
state 0.0000,0.0008,-0.0010,0.0069,-0.0016,-0.0100,-4.6161,
     -0.5956,4.6127,0.0025
set dispfile luxo.out
sim 8.0
quit
```

C.2 Scripts for Pogo

pogo.anix

```
aniset winx 600
aniset winy 600
objname base
path
        pts -0.02 0.02 0
        pts 0.52 0.02 0
        pts 0.52 -0.02 0
        pts -0.02 -0.02 0
        pts 0.00 0.08 0
        pts -0.15 0.03 0
        pts -0.05 0.0 0
        pts -0.15 -0.03 0
        pts 0.00 -0.08 0
objname leg
path
        pts -0.02 0.02 0
        pts 0.27 0.02 0
        pts 0.27 -0.02 0
        pts -0.02 -0.02 0
instance base link1
instance leg link2
instance leg link3
instance leg link4
instance leg link5
attach link1 to world
attach link2 to link1
attach link3 to link2
attach link4 to link1
attach link5 to link4
tf trans link2 0.0 0.0 0
tf trans link3 0.25 0.0 0
tf trans link4 0.50 0.0 0
tf trans link5 0.25 0.0 0
< gnd.anix
init
tf trans link1 3.5 0.0 0
tf trans link1 _1 _2 0
tf rot link1 Z _3
tf rot link2 Z _4
tf rot link3 Z _5
tf rot link4 Z _6
tf rot link5 Z _7
tf trans world 14 6 0
tf scale world 3 3  1
aniset degmode f
aniset erase t
```

pogo.desc

```
link 1 0 0.0 0.0 0.15 0.003123 0.25 0
link 2 1 0.0 0.0 0.10 0.002082 0.125 0
link 3 2 0.25 0.0 0.10 0.002082 0.125 0
link 4 1 0.50 0.0 0.10 0.002082 0.125 0
link 5 4 0.25 0.0 0.10 0.002082 0.125 0
monitor 3 1 0.27 0
monitor 3 2 -0.02 0
monitor 5 3 0.27  0
monitor 5 4 -0.02  0
set procname pogo
compile
quit
```

pogo.sim

```
set kdamp 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
set dtsim 0.001
dyn pogo
mon mon_pogo 4
set state_size 14
state 0.0000,0.2487,0.3512,-0.4275,-0.1337,-1.7569,-1.7014,
      0.2469,5.2939,0.0589,-1.1427,-0.3886,4.4908,-0.5539
set dispfile pogo.out
sim 8.0
```

C.3 Scripts for Walker

walker.anix

```
aniset winx 600
aniset winy 600
objname base
path
        pts -0.02 0.02 0
        pts 0.52 0.02 0
        pts 0.52 -0.02 0
        pts -0.02 -0.02 0
objname leg
path
        pts -0.02 0.02 0
        pts 0.27 0.02 0
        pts 0.27 -0.02 0
        pts -0.02 -0.02 0
objname foot
path
        pts -0.082 0.02 0
        pts 0.082 0.02 0
        pts 0.082 -0.02 0
        pts -0.082 -0.02 0
instance base link1
instance leg link2
instance leg link3
instance foot link4
instance leg link5
instance leg link6
instance foot link7
attach link1 to world
attach link2 to link1
attach link3 to link2
attach link4 to link3
attach link5 to link1
attach link6 to link5
attach link7 to link6
tf trans link2 0.0 0.0 0
tf trans link3 0.25 0.0 0
tf trans link4 0.25 0.0 0
tf trans link5 0.0 0.0 0
tf trans link6 0.25 0.0 0
tf trans link7 0.25 0.0 0
< gnd.anix
init
tf trans link1 0.0 0.0 0
tf trans link1 _1 _2 0
tf rot link1 Z _3
tf rot link2 Z _4
tf rot link3 Z _5
tf rot link4 Z _6
tf rot link5 Z _7
tf rot link6 Z _8
tf rot link7 Z _9
tf trans world 10 6 0
tf scale world 4 4 1
aniset degmode f
aniset erase t
```

walker.desc

```
link 1 0 0.00 0.0  3.0 0.0625 0.250 0
link 2 1 0.00 0.0  5.0 0.0260 0.125 0
link 3 2 0.25 0.0  4.0 0.0208 0.125 0
link 4 3 0.25 0.0  5.0 0.0260 0.0 0
link 5 1 0.00 0.0  4.0 0.0208 0.125 0
link 6 5 0.25 0.0  1.0 0.0052 0.125 0
link 7 6 0.25 0.0  1.0 0.0052 0.0 0
monitor 4 1 -0.082 0.0
monitor 4 2 0.082  0.0
monitor 7 3 -0.082 0.0
monitor 7 4 0.082 0.0
set procname walker
compile
quit
```

walker.sim

```
set kdamp 0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
set dtsim 0.001
dyn walker
mon mon_walker 4
set state_size 18
state 0.0,0.0,0.52,0.0,1.120,0.0,-2.793,0.0,-0.175,0.0,
      1.745,0.0,-2.382,0.0,-0.885,0.0,1.745
set dispfile walker.out
sim 8.0
quit
```

Bibliography

[1] R. Alexander. The gaits of bipedal and quadrupedal animals. *The International Journal of Robotics Research*, 3(2):49–59, 1984.

[2] R. Alexander and A. Jayes. Vertical movements in walking and running. *Journal of Zoology*, 185:27–40, 1978.

[3] William Armstrong and Mark Green. The dynamics of articulated rigid bodies for purpose of animation. *Visual Computer*, 1(4):231–240, 1985.

[4] B. Arnaldi, G. Dumont, and G. Hegron. Animation of physical systems from geometric, kinematic and dynamic models. In *Proceedings IFIP, on Modelling in Computer Graphics*, April 1991.

[5] N. I. Badler, C. B. Phillips, and B. L. Webber. *Simulating Humans*. Oxford University Press, 1993.

[6] Norman I. Badler, Kamran H. Manoochehri, and Graham Walters. Articulated figure positioning by multiple constraints. *IEEE Computer Graphics and Applications*, 7(6):28–38, 1987.

[7] D. H. Ballard. *Computer vision*. Prentice-Hall, 1982.

[8] D. Baraff. Determining frictional inconsistency for rigid bodies is np-complete. Technical Report TR 90-1112, Cornell University, 1990.

[9] David Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *Computer Graphics, SIGGRAPH'90*, 24:19–28, 1990.

[10] R. Barzel. *Physically-Based Modelling for Computer Graphics*. Academic Press Inc., 1992.

[11] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constraints. *Computer Graphics*, 22(4):179–188, 1988.

[12] R. Beer, R. Ritzmann, and T. McKenna. *Biological neural networks in invertebrate neuroethology and robotics*. Academic press, 1993.

[13] R. Boulic, N. Thalmann, and D. Thalmann. A global human walking model with real-time kinematic personification. *The Visual Computer*, 6:344–358, 1990.

[14] R. A. Brooks. A robot that walks: Emergent behaviours from carefully evolved network. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making them move: mechanics, control and animation*, pages 209–232. Morgan Kaufmann, 1991.

[15] Lynne Brotman and Arun Netravali. Motion interpolation by optimal control. *Computer Graphics*, 22:309–315, August 1988.

[16] Armin Bruderlin and Thomas W. Calvert. Goal-directed dynamic animation of human walking. *Computer Graphics*, 23(3):233–242, 1989.

[17] J. Canny. Collision detection for moving polyhedra. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 8(2):200–209, 1986.

[18] R. T. Chin and C. R. Dyer. Model-based recognition in robot vision. *Computing Surveys*, 18(1):67–108, 1986.

[19] C. K. Chow and D. H. Jacobson. Studies of human locomotion via optimal programming. *Mathematical Bioscience*, 10:239–306, 1971.

[20] Michael F. Cohen. Interactive space time control for animation. *Computer Graphics*, 26:293–302, July 1993.

[21] N. Corby and J. Mundy. Applications of range image sensing and processing. In R. C. Jain and A. K. Jain, editors, *Analysis and interpretetion of range images*, pages 273–337. Springer-Verlag, 1990.

[22] J. J. Craig. *Introduction to robotics mechanics and control.* Addision-Wesley, 1986.

[23] James Cremer. *An architecture for general purpose physical system simulation–integratinggeometry, dynamics, and control.* PhD thesis, Cornell University Ithaca, New York, 1989.

[24] L. Davis. *Handbook of genetic algorithms.* Van Nostrand Reinhold, 1991.

[25] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Journal of Applied Mechanics*, pages 215–221, June 1955.

[26] T. Duff. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *Computer Graphics, SIGGRAPH'92*, 26:131–138, 1992.

[27] Geist et. al. *PVM 3 user's guide and reference manual.*

[28] D. Fogel. Asymptotic convergence properties of genetic algorithms and evolutionary programming: analysis and experiments. *Cybernetics and Systems: An International Journal*, 25:389–407, 1994.

[29] J. D. Foley, A. V. Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics.* Addison-Wesley, 1990.

[30] A. Fukunaga, L. Hsu, P. Reiss, A. Shuman, J. Christenen, J. Marks, and J. T. Ngo. Motion-synthesis techniques for 2d articulated figures. Technical Report TR-05-94, Harvard University, Center for Research in Computing Technology, 1994.

[31] R. Full. Integration of individual leg dynamics with whole body movement in arthropod locomotion. In R. Beer, R. Ritzmann, and T. McKenna, editors, *Biological neural networks in invertebrate neuroethology and robotics*, pages 3–20. Academic Press, 1993.

[32] P. Gill, W. Murray, and M. Wright. *Practical optimization.* Academic Press, New York, 1981.

[33] Michael Girad. Interactive design of 3d computer-animated legged animal motion. *IEEE Computer Graphics and Applications*, 7(6):39–51, June 1987.

[34] Michael Girad. Constrained optimization of articulated body in computer animation. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making them move: mechanics, control and animation*, pages 209–232. Morgan Kaufmann, 1991.

[35] Michael Girad and A. A. Maciejewski. Computational modelling for the computer animation of legged figures. *Computer Graphics*, 19:263–270, 1985.

[36] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, 1989.

[37] L. Gritz and J. Hahn. Genetic programming for articulated figure motion. *The Journal of Visualization and Computer Animation*, 6(3):129–142, 1995.

[38] R. Grzeszczuk and D. Terzopoulous. Automated learning of muscle-actuated locomotion through control abstraction. *Computer Graphics, SIGGRAPH'95*, 29:63–70, 1995.

[39] J. K. Hahn. Realistic animation of rigid bodies. *Computer Graphics, SIGGRAPH'88*, 22(4):299–308, 1988.

[40] S. Hansen, J. Kearney, and J. Cremer. Motion control through communicating hierarchical state machines. *Fifth Eurographics workshop on Animation and Simulation*, September 1994.

[41] R. Harelick and L. Shapiro. *Computer and robot vision, Vol. II*. Addision-Wesley, 1993.

[42] E. Haug. *Computer aided analysis and optimization of mechanical system dynamics. NATO ASI Series F: Computer and System Sciences, Vol.9*. Springer-Verlag, 1984.

[43] M. Hebert, T. Kanade, and I. Kweon. 3-d vision technique for autonomous vehicles. In R. C. Jain and A. K. Jain, editors, *Analysis and interpretation of range images*, pages 273–337. Springer-Verlag, 1990.

[44] V. Herzen and A. Barr H. Zatz. Geometric collision for time-dependent parametric surfaces. *Computer Graphics, SIGGRAPH'90*, 24(4):39–48, 1990.

[45] M. Hildebrand. Symmetrical gaits of horse. *Science*, 150:701–708, 1965.

[46] E. C. Hildreth. *The measurement of visual motion*. MIT press, 1983.

[47] Victor Ng Thow Hing. A biomechanical musculotendon model for animating articulated objects. Master's thesis, University of Toronto, 1994.

[48] Victor Ng Thow Hing. Research issues in the design of control methods for physically-based computer animation. Unpublished, 1995.

[49] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien. Animating human athletics. *Computer Graphics, SIGGRAPH'95*, 29, 1995.

[50] Jessica Hodgins and Mark Raibert. Biped gymnastics. *The International Journal of Robotics Research*, 9(2):115–132, 1990.

[51] J. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, 1975.

[52] Paul M. Issac and Michael F. Cohen. Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics. *Computer Graphics, SIGGRAPH'87*, 21:215–224, July 1987.

[53] V. V. Kamat. A survey of techniques for simulation of dynamic collision detection and response. *Computers and Graphics*, 17(4):379–385, 1993.

[54] V. V. Kamat. Synthesis of realistic motion for legged creatures. *Software Bulletin*, 3(4):9–16, 1995.

[55] Michael Kass. Inverse problems in computer graphics. In N.M.Thalmann and D.Thalmann, editors, *Creating and animating the virtual world*, pages 21–33, Springer-Verlag, 1992.

[56] J. A. S. Kelso and A. S. Pandya. Dynamic pattern generation and recognition. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making them move: mechanics, control and animation*, pages 171–190. Morgan Kaufmann, 1991.

[57] D. E. Kirk. *Optimal Control Theory: An Introduction*. Prentice-Hall Inc., 1970.

[58] S. Kirkpatric, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(13):671–680, May 1983.

[59] H. Ko and N. I. Badler. Stright line walking animation based on kinematic generalization that preserves original characteristics. *Proceedings of Graphics Interface'93*, pages 9–16, May 1993.

[60] D. H. U. Kochanek and R. H. Bartels. Interpolating splines with local tension, continuity and bias control. *Computer Graphics, SIGGRAPH'84*, 18(3):33–41, 1984.

[61] B. Lafleur, M. Thalmann, and D. Thalmann. Cloth animation with self-collision detection. In *Animation of Synthetic Actors and 3D Interaction*, 1991.

[62] S. Levy. *Artificial life: the quest for a new creation*. Pantheon Books, New York, 1992.

[63] M. C. Lin and J. Canny. Efficient collision detection for animation. *Third Eurographics Workshop on Animation and Simulation*, September 1992.

[64] Zicheng Liu, Steven J. Gortler, and Michael F. Cohen. Hierarchical space-time control. *Computer Graphics*, 28:35–42, 1994.

[65] P. Lötstedt. Numerical simulation of time-dependent contact friction problem in rigid body. *SIAM Journal of Scientific Statistical Computing*, 5(2):370–393, 1984.

[66] T. Lozano-Pérez and M. Wesley. An algorithm for planning collision free paths among polyhedral obstacles. *Communication of ACM*, 22:560–570, 1979.

[67] D. Luenberger. *Introduction to linear and nonlinear programming*. Addison-Wesley, 1973.

[68] P. Maes and R. Brooks. Learning to coordinate behaviours. *Proc. of AAAI'90*, pages 796–802, 1990.

[69] T. McGeer. Passive dynamic walking. *The International Journal of Robotics Research*, 9(2):62–82, 1990.

[70] R. B. McGhee. Some finite state aspects of legged locomotion. *Mathematical Bioscience*, 2:67–84, 1968.

[71] M. McKenna and D. Zeltzer. Dynamic simulation of autonomous legged locomotion. *Computer Graphics SIGGRAPH'90*, 24:29–38, August 1990.

[72] T. McMahon. Mechanics of locomotion. *The International Journal of Robotics Research*, 3(2):5–26, 1984.

[73] D. Metaxas and D. Terzopoulos. Dynamic deformation of solid primitives with constraints. *Computer Graphics, SIGGRAPH'92*, 26(2):309–312, 1992.

[74] Z. Michalewicz. *Genetic algorithms + Data structure = Evolution programs*. Springer-Verlag, 1992.

[75] Mathew Moore and Jane Wilhelms. Collision detection and response for computer animation. *Computer Graphics*, 22(4):289–298, 1988.

[76] M. Mortenson. *Geometric modelling*. Wiley, New York, 1985.

[77] S. P. Mudur and P. A. Koparkar. Interval methods for processing geometric objects. *IEEE Computer Graphics and Applications*, 4(7):7–17, 1984.

[78] S. P. Mudur and J. H. Singh. A notation for computer animation. *IEEE Transaction Systems, Man, and Cybernetics*, SMC-8(4):308–311, 1978.

[79] Magnenat-Thalmann N. and D. Thalmann. *Computer animation: theory and practice*. Springer-Verlag, 1985.

[80] J. T. Ngo and J. Marks. Physically realistic motion synthesis in animation. *Evolutionary Computation*, 1(3):235–268, 1993.

[81] J. Thomas Ngo and Joe Marks. Spacetime constraints revisited. *Computer Graphics, SIGGRAPH'93*, 27:344–350, August 1993.

[82] P. E. Nikravesh. *Computer-aided analysis of mechanical systems*. Prentice-Hall, 1988.

[83] S. Parry-Barwick and A. Bowyer. Is the feature interface ready? In R. Martin, editor, *Directions in geometric computing*, pages 129–160. Information Geometers, 1993.

[84] T. Pavlidas. *Structural pattern recognition*. Springer-Verlag, 1977.

[85] F. P. Preparata and M. I. Shamos. *Computational geometry, an introduction*. Springer-Verlag, 1985.

[86] M. Raibert and J. Hodgins. Legged robots. In R. Beer, R. Ritzmann, and T. McKenna, editors, *Biological neural networks in invertebrate neuroethology and robotics*, pages 319–354. Academic Press, 1993.

[87] Mark Raibert and Jessica Hodgins. Animation of dynamic legged locomotion. *Computer Graphics*, 25:349–358, July 1991.

[88] L. Scales. *Introduction to non-linear optimization*. Macmillan, 1985.

[89] Karl Sims. Evolving virtual creatures. *Computer Graphics*, 28:15–22, July 1994.

[90] A. Smith. Tutorial notes: Introduction to computer animation. *Spline tutorial notes- Technical memo No. 77, SIGGRAPH'83*, pages 64–75, July 1983.

[91] B. Snider. The toy story. *Wired*, December 1995.

[92] J. Snyder. Interval methods for multi-point collision between time-dependent curved surfaces. *Computer Graphics, SIGGRAPH'93*, 27, 1993.

[93] M. Srinivas and L. M. Patnaik. Genetic algorithms: a survey. *IEEE Computer*, 27(6):17–26, 1994.

[94] S. N. Steketer and N. I. Badler. Parametric keyframe interpolation incorporating kinetic adjustment of phrasing control. *Computer Graphics, SIGGRAPH'85*, 19(3):255–262, 1985.

[95] A. James Stewart and James F. Cremer. Animation of 3d human locomotion: climbing stairs and descending stairs. *Third Eurographics Workshop on Animation and Simulation*, September 1992.

[96] D. Terzopolos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. *Computer Graphics, SIGGRAPH'87*, 21(4):205–214, 1987.

[97] S. Ullman. *The interpretation of visual motion*. MIT press, 1979.

[98] Michiel van de Panne. anix an animation server for x–windows. Technical report, Dynamic Graphics Project, Dept. of Computer Science, University of Toronto, ftp site: dgp.utoronto.ca, in pub/van, 1989.

[99] Michiel van de Panne. Dynamics compiler, v1.1. Technical report, Dynamic Graphics Project, Dept. of Computer Science, University of Toronto, ftp site: dgp.utoronto.ca, in pub/van, 1989.

[100] Michiel van de Panne and Eugene Fiume. Sensor-actuator networks. *Computer Graphics, SIGGRAPH'93*, 27:335–342, August 1993.

[101] Michiel van de Panne, Eugene Fiume, and Zvonko Vranesic. Reusable motion synthesis using state-space controller. *Computer Graphics SIGGRAPH'90*, 24:225–234, August 1990.

[102] Michiel van de Panne, Ryan Kim, and Eugene Fiume. Synthesizing parameterized motions. *Fifth Eurographics workshop on Animation and Simulation*, September 1994.

[103] Michiel van de Panne, Ryan Kim, and Eugene Fiume. Virtual wind-up toys for animation. In *Proceedings of Graphics Interface*, pages 208–215, 1994.

[104] C. Walnum. *Adventures in artificial life*. Que corporation, 1993.

[105] Jane Wilhelms. Toward automatic motion control. *IEEE Computer Graphics and Applications*, 7(4):11–22, 1987.

[106] D. Winter. *Biomechanics and motor control of human movement*. John Wiley & Sons, 1990.

[107] J. M. Winters and S. Woo. *Multiple Muscle System: Biomechanics and Movement organization*. Springer-Verlag, 1990.

[108] Andrew Witkin and Michael Kass. Spacetime constraints. *Computer Graphics*, 22:159–168, August 1988.

[109] D. Zeltzer. Motor control techniques for figure animation. *IEEE Computer Graphics and Applications*, 2(9):53–59, November 1982.

[110] D. Zeltzer. Task-level graphical simulation: abstraction, representation and control. In Norman I. Badler, Brian A. Barsky, and David Zeltzer, editors, *Making them move: mechanics, control and animation*, pages 171–190. Morgan Kaufmann, 1991.