

**Locomotion of Jointed Figures
over Complex Terrain**

by

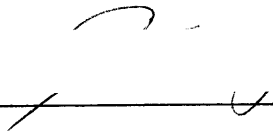
Karl Sims

Bachelor of Science in Life Sciences
Massachusetts Institute of Technology
Cambridge, Mass. 1984

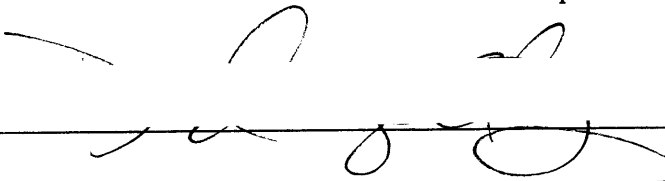
SUBMITTED TO THE DEPARTMENT OF ARCHITECTURE
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE
DEGREE
MASTER OF SCIENCE IN VISUAL STUDIES AT THE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUNE 1987

© Massachusetts Institute of Technology 1987


Signature of the Author


Karl Sims
Department of Architecture
May 8, 1987

Certified by


David Zeltzer
Thesis Supervisor
Assistant Professor of Computer Graphics

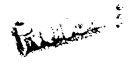
Accepted by


Nicholas Negroponte
Chairman
Departmental Committee on Graduate Students

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

JUN 08 1987

LIBRARIES



Locomotion of Jointed Figures over Complex Terrain

by

Karl Sims

Submitted to the Department of Architecture on May 8, 1987 in partial fulfillment of the requirements of the degree of Master of Science.

Abstract

This thesis describes a system that allows the creation of arbitrary jointed figures that can be animated traveling over uneven terrain with forbidden zones. Creatures can be easily described that have any number of legs, each leg having any number of joints, using an interactive, graphical "figure editor". Locomotion can be automatically generated from a figure description using default locomotion parameters, and these parameters can be adjusted to achieve a variety of locomotion styles. Techniques such as inverse-kinematics, dynamic simulation, step planning, and trajectory planning have been used.

Thesis Supervisor: David Zeltzer

Title: Assistant Professor of Computer Graphics

This work was supported in part by NHK (Japan Broadcasting Corporation), and by an equipment loan from Symbolics, Inc.

Contents

1	Introduction	4
1.1	Towards Efficient Expression of Visual Imagination	4
1.2	Feedback Time	5
1.3	Efficient Representation	6
1.4	Overview of Thesis	8
2	Relevant Work	10
3	Levels of Motion Representation	14
3.1	Controlling a Single Object	14
3.1.1	Position	14
3.1.2	Velocity	15
3.1.3	Force	15
3.2	Controlling a Group of Objects	16
3.2.1	Object Hierarchies	17
3.2.2	Limb Control	17
3.2.3	Locomotion control	18
3.2.4	Behavior and Goal Directed Control	19
4	Jointed Figure Editor	21
4.1	Interactive Environment	21
4.2	Network Analysis	24
5	Jointed Figure Representation	27
5.1	Joints and Links	27
5.2	Axis and Position Joint Representation	28
5.3	Joint Link Network	31

<i>CONTENTS</i>	4
5.4 Calculating the Link Positions and the Jacobian Matrix . . .	33
6 Inverse Kinematics	37
6.1 Pseudo-inverse Iterations	38
6.2 Preferred Joint Values	41
6.3 Dealing With Singularities	42
6.3.1 Allowing Full Extension	43
7 Dynamics	46
7.1 Dynamic Simulation of Single Rigid Bodies	47
7.1.1 Calculating Moment of Inertia	50
7.2 Dynamic Simulation of Articulated Figures	51
7.3 Generating Muscle Forces	52
7.3.1 Inverse Dynamics	52
7.3.2 Minimum Energy	52
7.4 Hybrid Dynamic-Kinematic Control	53
8 Terrain	55
8.1 Fast Surface Height Detection	55
8.2 Forbidden Zone Detection	57
9 Locomotion	59
9.1 Locomotion Parameters	59
9.2 Gaits	61
9.3 Planning Trajectories	63
9.4 Not Planning Trajectories	63
10 Walking	65
10.1 Body Path	66
10.2 Feet Trajectories: Step Planning	66
10.3 Automatic Gait Assignment	68
11 Pronking	70
11.1 Flight Phase	72
11.2 Landing Phase	73
11.3 Pushing Phase	75

<i>CONTENTS</i>	5
12 Bouncing	78
12.1 Bouncing in One Dimension	78
12.2 Simulating Bounces with Surface Forces	80
12.3 Bouncing in Three Dimensions: A Model for Polyhedral Ob- ject Bouncing	81
13 Sound	84
13.1 Sound Events	84
13.1.1 Collisions	85
13.1.2 Sounds from Motion	86
14 Conclusion	87
A Miscellaneous	89
A.1 Jointed Figure Editor Details	89
A.2 Shape Enclosure Test	91
A.3 Rotating about an Arbitrary Angle	92
B The Pseudo-inverse	93
B.1 The Jacobian	94
B.2 Secondary Goals	96
B.3 Combining Goals	99
C Acknowledgments	101

Chapter 1

Introduction

1.1 Towards Efficient Expression of Visual Imagination

Someday, computers should remove the non-creative human work from the process of creating animation. Because of the large amount of information in a moving image, the process between an animator's visual imagination and a physical visual experience that others can perceive is a difficult one. Computers have begun to provide useful tools for generating animation, but the tools are limited in many ways. A considerable time investment is still required by the animator, and the resulting animation is usually affected by the tool: computer animation looks like computer animation [65]. It is still difficult to create structure and motion of arbitrary quality and complexity without a fair amount of tedious work.

There are two fundamental ways the efficiency of a computer animation system can be improved: it can be faster, and it can be more intelligent.

1.2 Feedback Time

Speed is an obvious desirable quality of an animation system. The feedback time in which an animator sees results is critical. Fast interactive tools are important for specifying individual parameters of an animation such as: creating, positioning, and coloring individual objects, choosing lighting, and creating camera and object movements. It is also valuable to view the complete animation with a minimum delay.

It may be interesting to briefly compare visual communication and auditory communication with respect to feedback time. When a musician plays an instrument the result of his actions are heard instantaneously. This real time feedback allows the musician to quickly zero in on the sound he has in mind and play with “feeling”. Music synthesis is difficult when there is a delay between choosing parameters and hearing the resulting sounds, such as in most computer music. We don’t think of the possibility of creating arbitrary animation of complex moving objects in real time because it seems impossible, although we do participate in some real time visual communications such as dance.

The intelligence of an animation system also affects the feedback time. Efficient representation should decrease the time in the part of the feedback cycle that directly involves the animator.

1.3 Efficient Representation

There are many ways in which an animation system can be more intelligent. An animation system needs to understand things about physics and behavior, and should be able to communicate well with the animator. Tools for generating animation from abstract descriptions should be available. An animator should be able to describe structure and motion by specifying a small but efficient amount of information.

The ability to combine and abstract is essential. We need to build layers of procedures that allow increasingly higher levels of control. To efficiently represent new pieces of animation, it should be possible to make the animation system understand new representations. If new tools can easily be made by combining existing tools, the difference between building an animation system and using it narrows.

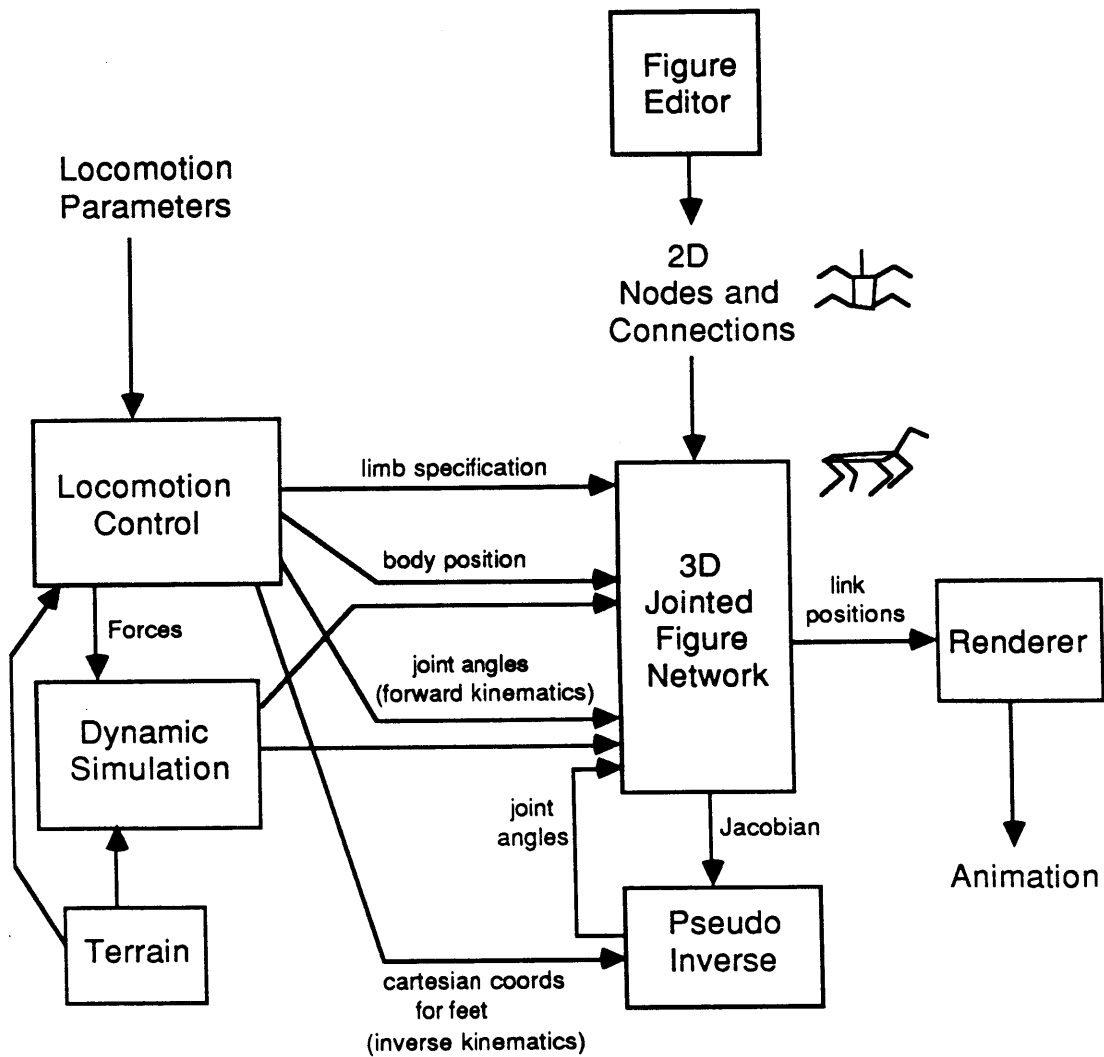
An example of efficient representation of structure is the fractal. Detailed surface shape can be described using only a few parameters, although the ability to specify the exact final surface is then lost.

Most of the work described in this thesis is concerned with efficient representation of motion. A system has been built that allows animation of locomotion of jointed figures, where the user provides only high level instructions instead of specifying many joint angles or position details for individual body parts.

A user can quickly draw an arbitrary creature with an interactive figure editor, this creature can become three dimensional and can then walk

or hop over uneven terrain completely automatically. No motion specification is necessary from the user, although locomotion parameters can be adjusted to vary the motion. The modules that work together to provide this functionality are outlined in figure 1.1.

figure 1.1



1.4 Overview of Thesis

Chapter 2 outlines relevant research that has been done in the fields of both computer graphics and robotics.

Chapter 3 explains how levels of motion representation can be built up to create complex motions for locomotion.

Chapter 4 describes a tool that allows interactive creation of jointed figures, and explains how they are analyzed to determine which parts are which.

Chapter 5 describes the method used to represent jointed figures such that forward and inverse kinematics can be easily performed on them.

Chapter 6 explains the details of techniques used to perform inverse-kinematics on redundant limbs. Details on the pseudo-inverse are included in Appendix B.

Chapter 7 discusses dynamic considerations for computer animation. The techniques used for dynamic simulation and hybrid dynamic kinematic control are described.

Chapter 8 explains how the terrain model is used for surface height detection and forbidden zone detection.

Chapter 9 discusses locomotion in general, and parameters such as gait that can be used to vary locomotion.

Chapters 10 and 11 explain the techniques used to generate animation of walking and pronking creatures over complex terrain.

Chapter 12 explains the techniques used to generate animation of bouncing polyhedra.

Chapter 13 discusses the animation systems ability to record sound events that can be used to create a sound track for the animation.

Finally, Chapter 14 summarized the animation system that has been implemented.

The Appendix contains some details on the jointed-figure editor user interface, some miscellaneous algorithms, and a section on the pseudo-inverse.

Chapter 2

Relevant Work

Many methods have been developed for describing and calculating motions of jointed figures from research in the fields of both robotics and computer graphics. See [77] for a survey of general techniques used in computer animation systems. Motion control for physical robots needs to be accurate and performed in real time. This currently often restricts locomotion of robots to be either simple or slow. Computer graphics animation does not suffer from the same restrictions as robotics, it does not need to be performed in real time and it can be less accurate, although it should still look correct. Some of the same techniques that are useful for generating and simulating robot motion are used in computer graphics animation.

Michael Girard has developed a legged figure animation system called PODA [13,14]. His work is probably the most related of any to the work described in this thesis. PODA uses key frame interpolation, inverse kinematics, and trajectory planning with dynamic considerations to generate almost realistic looking locomotion. The dynamic equations are decom-

posed into horizontal and vertical components to lower their complexity.

Girard's work and this work both involve generating motions for jointed figure locomotion. The main advantage of PODA over this system is that it allows a wider variation of movements to be created: figures have been animated turning, banking, and even dancing. The tradeoff is that PODA requires many more details for limb and body motions to be supplied by the animator than this system does, the animator creates postures and posture sequences to carefully build the motions for correct looking locomotion [14]. This system will automatically generate locomotion from a jointed figure description and a few locomotion parameters that can then be adjusted by the animator. PODA has generated locomotion only over flat terrain, whereas this system considers uneven terrain with forbidden footstep zones. PODA does not use dynamic simulation as this system can, although dynamically correct trajectories can be calculated.

David Zeltzer has done significant research on goal-directed animation systems [75,76,77,78]. A walking skeleton named George has been animated in which gait control parameters are adjusted as the environment changes so George can walk up and down slopes [76]. In [77] Zeltzer discusses a three level hierarchy for character animation: guiding, animator level, and task level. Guiding includes motion recording, key-frame interpolation, and shape interpolation systems. Animator level systems allow algorithmic specification of motion. Task level animation systems must contain knowledge about the objects and environment being animated, and the execution of motor programs is organized by the animation system.

The work described in this thesis is an example of one component of a task level animation system.

Marc Raibert has developed physical 2D one-legged and two-legged hopping machines, as well as 3D one-legged and four-legged hopping machines [48,49,50,51,52,53]. The 3D one-legged hopper can run and balance on a flat surface [49]. The control of the machine is decomposed into parts: forward running velocity is controlled by the position of the foot with respect to the center of gravity, body attitude is controlled by torquing the leg during support phase, and hopping height is controlled by the amount of thrust exerted by the leg on each hop.

Norman Badler has generated animation of human figures using a system called TEMPUS [4,5,6,7]. A biped called "bubble woman" has been animated performing various motions. Badler does not use inverse-kinematics in his work, the abilities of abstraction and adaptive motion are limited, and this system is restricted to human figures.

Jane Wilhelms has done some analysis on dynamic simulation for realistic animation of articulated bodies [72]. She has animated falling human bodies and single arms, but the dynamic simulation is computationally expensive and has prevented the use of dynamic simulation with complex articulated figures.

W. W. Armstrong and M. Green have developed a solution to the dynamic equations of motion for computer animation that grows linearly in computation time with the number of links [2]. They also discuss a technique for developing human figure models based on these dynamics.

Anthony Maciejewski, [28,29,13], and Charles Klein, [24,25,26,29,37], have done research on using the redundancy in robot manipulator arms. Pseudo-inverse control allows the redundancy of limbs to be used for achieving secondary goals such as obstacle avoidance [24,26,28,29,37]. They have also worked on some methods for generating motions of legged structures [25,13].

Chapter 3

Levels of Motion Representation

There are many ways to control the motions of objects. Single objects are fairly easy to control, but when objects start affecting each other and have constraints and dependencies, the problem becomes more difficult. Primitive tools for generating motion of single objects are combined and abstracted to give tools for creating motions for groups of objects. High levels of control can eventually be created that allow complex motions such as locomotion of jointed figures.

3.1 Controlling a Single Object

3.1.1 Position

Position control is a primitive tool in any 3d modeling or animation system. A single object may have nine degrees of freedom, translation, rotation, and scaling, on each of the three axes. Keyframe interpolation and scripting are methods that allow generating motion from a series of positions. Spline

curves are usually used to smoothly connect the user specified positions [71,77].

3.1.2 Velocity

Velocity control is not commonly used in animation systems. Some flight simulators use velocity control for moving the camera position, although acceleration or force control is probably a more accurate model of a real airplane. Given velocity specifications for an object the change in position can be found by integrating velocity. When using velocity control, the animator may begin to lose the ability to predict the resulting positions of the object.

3.1.3 Force

In reality, forces are actually what cause change in motion. It is impossible to just tell an object to be at a certain location as we can in the imaginary world of computer graphics. In reality we must apply forces to move something, although it is amazing that when we move ourselves we can think of it in terms of position and the correct forces seem to occur unconsciously. Force control is equivalent to acceleration control except the mass has an effect in force control. Force is directly related to acceleration by the relationship $F = ma$. An acceleration causes a change in velocity which in turn causes a change in position.

Given the positions, velocities, or accelerations of an object, the other two can easily be determined by differentiating or integrating. We often

care where objects end up, which can be difficult to predict using force control, but because force control produces more realistic motion it is often worthwhile. The motion of an object under the influence of gravity may be easier to specify with force control than position control, since gravity is a constant acceleration. Muscles forces have strict maximums, which are easier to avoid using force control. Muscle forces require energy that might want to be minimized. A discontinuous velocity would require an unrealistic pulse force of zero duration, so force control assures a path of an object will have a continuous derivative and a more realistic dynamic motion.

3.2 Controlling a Group of Objects

Often, desirable animation involves a large number of solid objects. There may be constraints on groups of objects, or they may have dependencies on each other. It becomes unreasonable for an animator to specify a motion for each object independently. It is necessary to have the ability to describe movements for an entire group of objects abstractly. An example of this that will be described later is specifying the movement of the sections of a leg by only specifying the movement of the foot. Other examples are describing the motion of a large number of particles or objects with a set of rules that is applied to each object. These techniques can give rise to effects such as fire, grass, flocking birds, or schooling fish [54,55,56,57].

3.2.1 Object Hierarchies

It is often useful to have objects move relative to other objects which may also have motion. For example it is easier to specify the movement of a head of a figure as a movement relative to its body. The animator specifies motion for each individual object relative to its parent, and the final motion of an object depends on the motions of all the objects above it in the hierarchy. This is a useful representation when objects are physically attached to each other in some way. But describing motion for each object still requires the same amount of information from the animator.

3.2.2 Limb Control

A limb will be any chain of solid links connected by joints. A limb will have a base joint that is probably connected to a body of a creature, and an end-effector that is probably a hand or foot. The motion of each link is constrained by the axes of the joint or joints that connect it to other links. Each link can be positioned relative to its parent by rotating a single joint. This is called forward kinematics. Notice that it can be difficult for the animator to predict where the end-effector of the limb will end up especially if there are a large number of joints. Many iterations of the human animator adjusting joint angles may be necessary before the desired motion is achieved.

Often a more desirable mechanism for positioning the links of a limb is to specify a position of the end-effector. A set of joint angles that will give

this desired end-effector position can be automatically calculated using a technique called inverse-kinematics that is described in detail in chapter 6. With inverse-kinematics tools the amount of information necessary to control a limb can be cut down by a significant amount. A limb with 6 joints could be controlled by just giving 3 position parameters for the end-effector as opposed to giving position and orientation for each link of the limb separately which corresponds to 36 parameters (6 joints x 6 degrees of freedom). The motion of an entire limb can often be efficiently described by specifying the motion of its end-effector.

3.2.3 Locomotion control

Given a description of a legged creature, the creature can be instructed to transport itself over a given terrain in a variety of locomotion styles. The motions of all the parts of the creature are generated automatically by the animation system from a set of locomotion parameters. Locomotion control combines other methods of control such as limb control and force control. Force control is used to help produce dynamically correct looking motions such as hopping. Inverse-kinematics is used to control the limbs from feet trajectories.

Locomotion should be adaptive to the terrain. Creatures can adjust their pitch and altitude as they walk over bumps in the terrain. Each foot of a walking creature should stay in contact with the the surface when the foot is used for support, and should step over obstructions in the terrain if necessary.

There are many ways a creature can travel over a certain surface. A set of locomotion parameters such as a gait pattern and speed can be changed to give a wide variety of possible locomotion styles.

3.2.4 Behavior and Goal Directed Control

In behavior and goal directed control, even more abstract information would be supplied by the animator. For example, a character might be given the following instructions: "Go to the center of the room and jump over the box", "pace back and forth impatiently", or "hop on one foot around the room". The information might be provided to the animation system in some way other than actual English but it would be equally efficient.

Since there are many ways to reach most physical goals, one of the ways could be chosen from a behavior specification. A simple walk for example can express a variety of feelings such as happy, sad, peaceful, or mad. Different moods could be mapped into different locomotion parameters, for example, happiness might increase the bounciness of the body trajectory. The animator would have the ability to view the default results for a given mood and then adjust the locomotion parameters to fine tune the style.

An even more intelligent animation system might be able to take information such as character descriptions including personalities, and an environment description, and generate animation from that. It would have to make decisions about how each character would behave and speak, and how they would interact with each other, and then generate the appropriate motions and sounds. The animator could instruct them but only on the

level that a play director might instruct his actors.

Before motion can be generated intelligently, the structures that the motions will be applied to must be created. The next chapter explains how jointed figures have been created for use in animation.

Chapter 4

Jointed Figure Editor

This chapter describes a system for interactively creating arbitrary two dimensional networks of nodes and connections. These networks are used to create three dimensional networks of joints and links that represent legged creatures that can be animated efficiently.

4.1 Interactive Environment

The interactive environment used for creating jointed figures consists of a menu of operations that can be chosen with a mouse, and a mouse sensitive area of the screen that allows drawing nodes and connections. Figures are created flat as if on a dissection table, with head up, tail down, and legs in any other direction. Below are four examples of figures that have been drawn with the jointed figure editor.

figure 4.1a

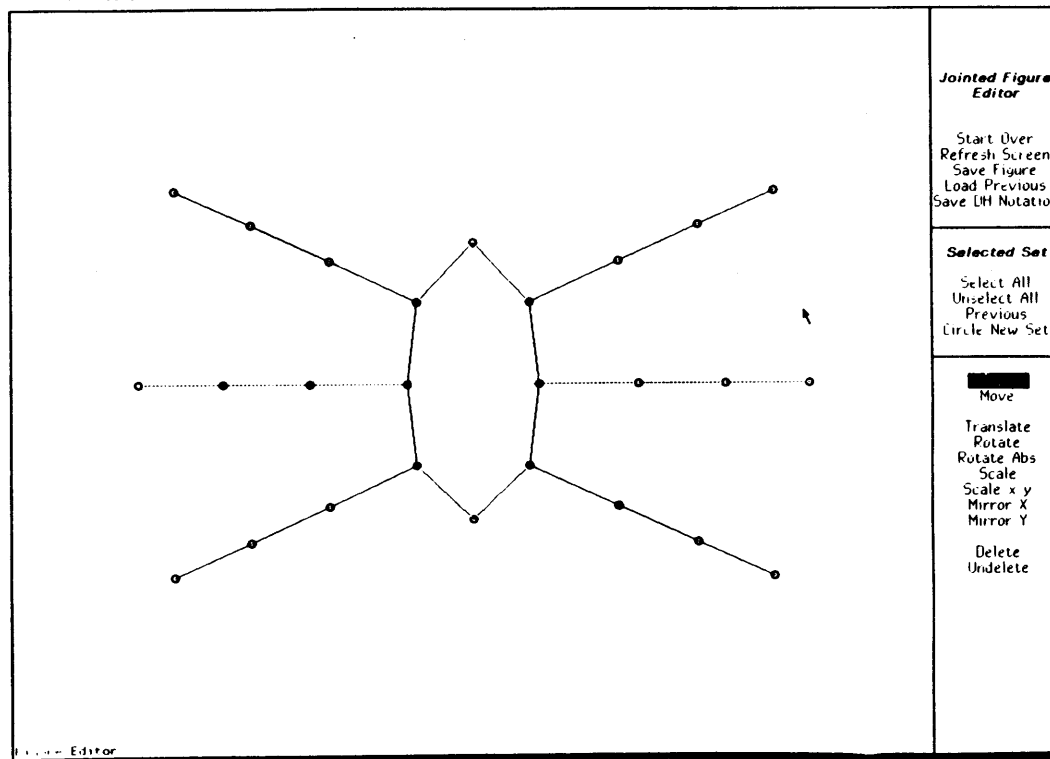
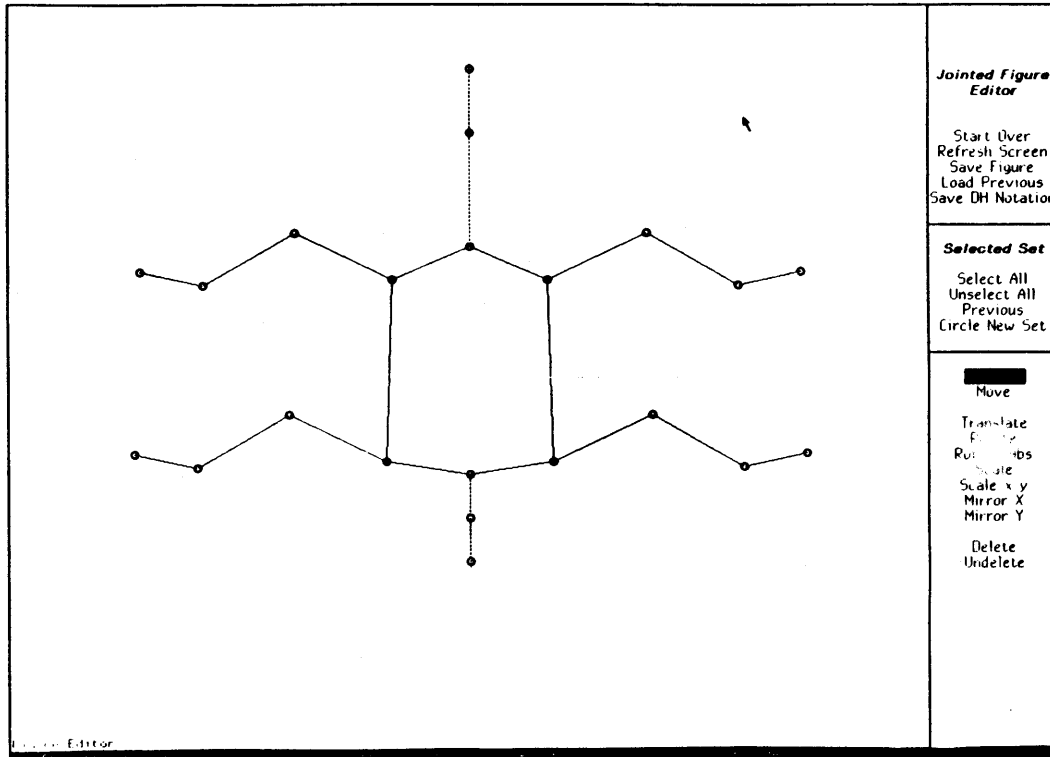
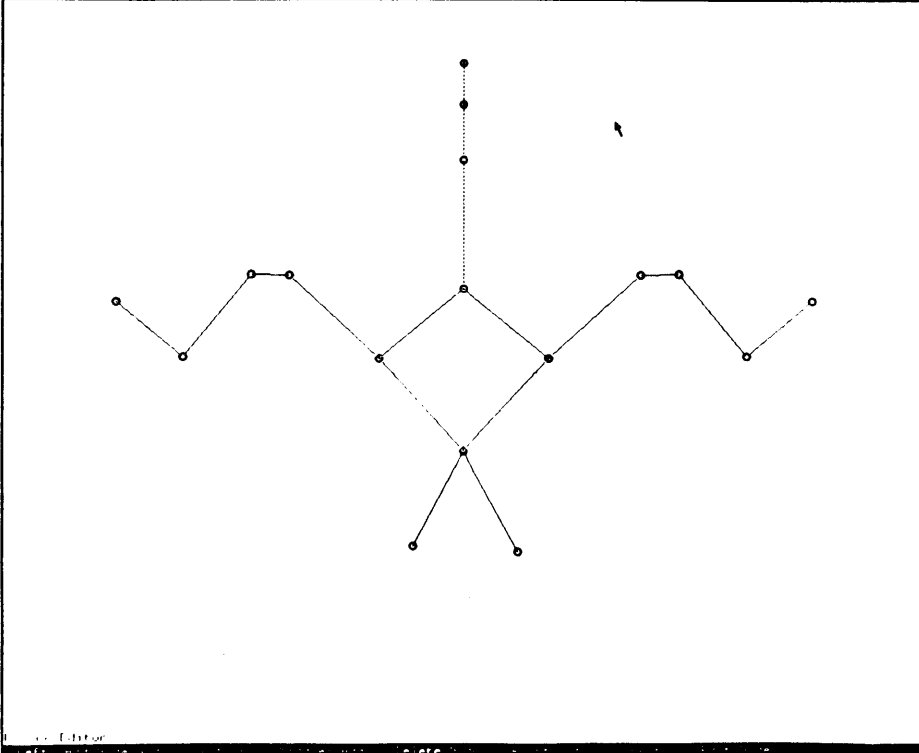
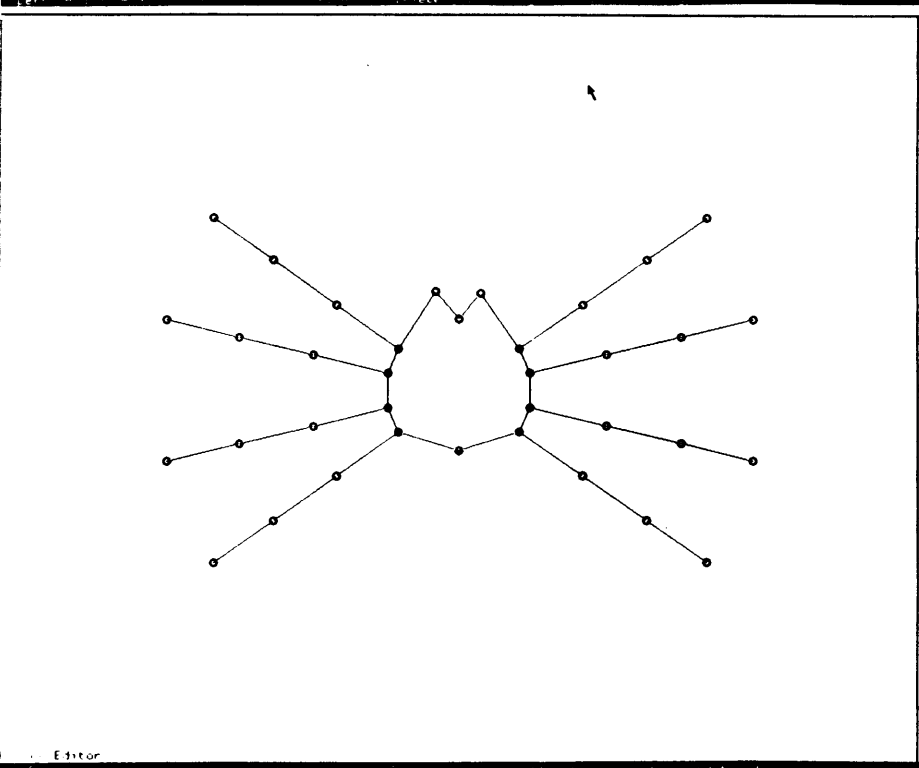


figure 4.1b

	<p>Jointed Figure Editor</p> <p>Start Over Refresh Screen Save Figure Load Previous Save DH Notation</p> <p>Selected Set</p> <p>Select All Unselect All Previous Circle New Set</p> <p>Move</p> <p>Translate Rotate Rotate Abs Scale Scale x y Mirror X Mirror Y</p> <p>Delete Undelete</p>
	<p>Jointed Figure Editor</p> <p>Start Over Refresh Screen Save Figure Load Previous Save DH Notation</p> <p>Selected Set</p> <p>Select All Unselect All Previous Circle New Set</p> <p>Move</p> <p>Translate Rotate Rotate Abs Scale Scale x y Mirror X Mirror Y</p> <p>Delete Undelete</p>

A set of operations that can move or copy sets of nodes allow arbitrary jointed figures to be created efficiently. Any pattern in the structure can be taken advantage of to save time. If the figure has symmetry or similar legs, part of the figure can be made and then duplicated, or mirrored. Details of the interactive environment for creating jointed figures are provided in the first section of the appendix.

4.2 Network Analysis

Once a two dimensional network of nodes and connections has been created using the tools briefly described above, the network is analyzed to determine what animal parts the nodes represent. The legs and body are determined, and the head and tail are found if they exist. Then 3-D joints and links are created and joint axes are assigned.

First the largest group of connected nodes is found, any nodes not connected to this group are ignored. Nodes with only one connection are assumed to be the appendage tips. These are each followed until a node with more than one connection is reached, this will be the base node of the appendage. In this way all appendages are determined. The appendages are separated into legs, head, and tail, depending on the position of their base nodes. The body of the figure is described by all of the base nodes of the appendages plus nodes that are not part of any appendage.

A three dimensional joint-link network is created from this two dimensional information. In figure-editor space, (x_f, y_f) , the head of the creature

is assumed to point up along the $-Y_f$ axis, in the 3D jointed-figure creation coordinate system, (x_c, y_c, z_c) , the head is assumed to point along the $+x_c$ axis, and the $-y_c$ axis is assumed to be up. The transformation from 2D figure-editor space to 3D jointed-figure space can be defined by the following matrix:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_f \\ y_f \end{bmatrix} \quad (4.1)$$

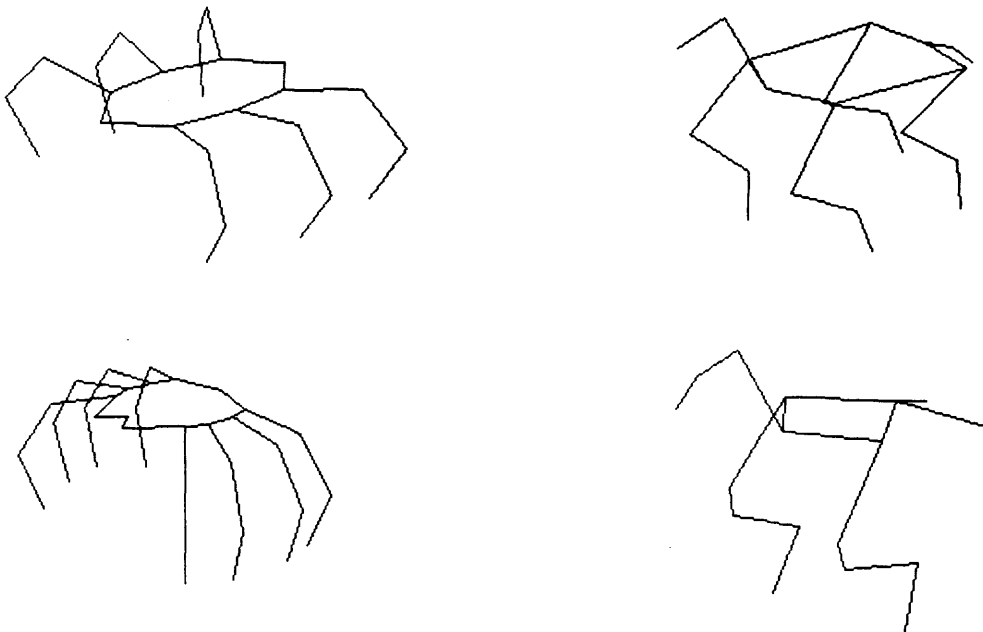
There is not always a one to one mapping between 2D node connections and 3D solid links. The body which is described by a group of nodes that are usually interconnected with several connections, becomes a single solid 3D link having joints connecting it to each appendage, although the rest of the connections that are not part of the body each produce a single link. There is also not always a one to one mapping between 2D nodes and 3D joints. In this work, each joint will be limited to one degree of freedom, either rotational or prismatic. Ball joints and Spherical joints can be created by combining more than one single-degree-of-freedom joint.

For the new creature to move its joints in three dimensions, axes of rotation must be determined for each 3D joint. Some nodes will need more than one axis of rotation, for example hips and shoulders become ball joints, which are actually multiple joints with one axis of rotation each.

There are two styles of creatures that can be made from the jointed figure editor: "insect" and "mammal". Each style determines a different set of default joint axes. These two choices for defaulting joint axes seem

to cover most cases fairly well. A variety of jointed figures have been made that are somewhat recognizable as realistic creatures. Spiders, beetles, water bugs, horses, camels, antelopes, kangaroos, and birds have all been created, as well as many imaginary unrecognizable creatures. “Insect” style causes joint axes to align perpendicular to the two links the joint connects. If we held our arms out to the sides with thumbs down and elbows up, this would mimic the “insect” style. “Mammal” style causes joint axes to be in the up direction, so that when the legs are folded down, the axes point out to the side and they bend correctly as our legs do. Ball joints are created at the base of every leg for both “insect” and “mammal” styles. Below are three dimensional examples of an “insect” and a “mammal” made with the jointed figure editor.

figure 4.2, “insects” and “mammals”



Chapter 5

Jointed Figure Representation

This chapter describes the data structures used to represent jointed figures and procedures that allow manipulation of them. Jointed figures are represented in such a way that forward kinematics or inverse kinematics can be performed between any two parts of the figure.

5.1 Joints and Links

The solid parts of a jointed figure will be referred to as links. Links are flexibly connected by joints. Joints and links mutually connect each other, but a joint can only connect two links, whereas a link can connect any number of joints. Circular connections, or closed chains are not permitted. Each joint has a single degree of freedom: it allows rotation or translation about a specific axis between the two links that it connects. Revolute joints rotate around the axis, prismatic joints slide along the axis. Joints describe the flexibility of the figure and determine the relative positions of the links. Links are the physical objects that a figure is actually made of. It is the

positions and orientations of the links that we are ultimately concerned with for rendering. In the illustrations that follow, joints will be drawn as circles, and links will be drawn as polygonal shapes or straight lines.

5.2 Axis and Position Joint Representation

As described in the previous chapter, a jointed figure can be created from a two dimensional network of nodes and connections made by an interactive figure editor. A network of joints and links could also be constructed procedurally. A procedure or the jointed figure editor produces a set of joints and links where each joint's data structure contains the following information:

1. Position in the creation coordinate frame.
2. Axis in the creation coordinate frame.
3. One or two links that the joint is attached to.

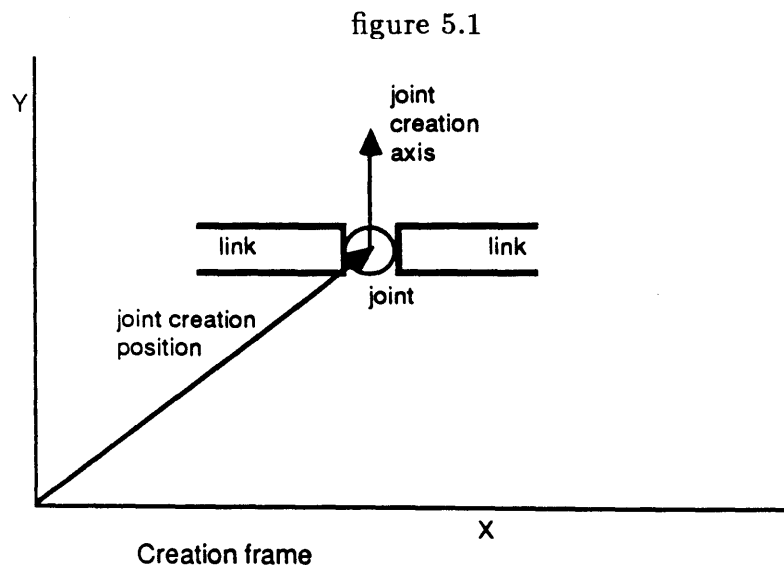
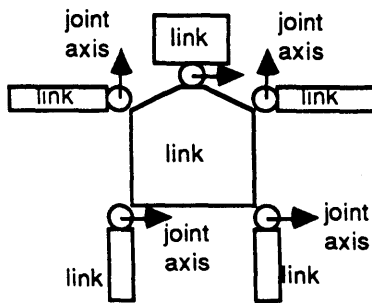
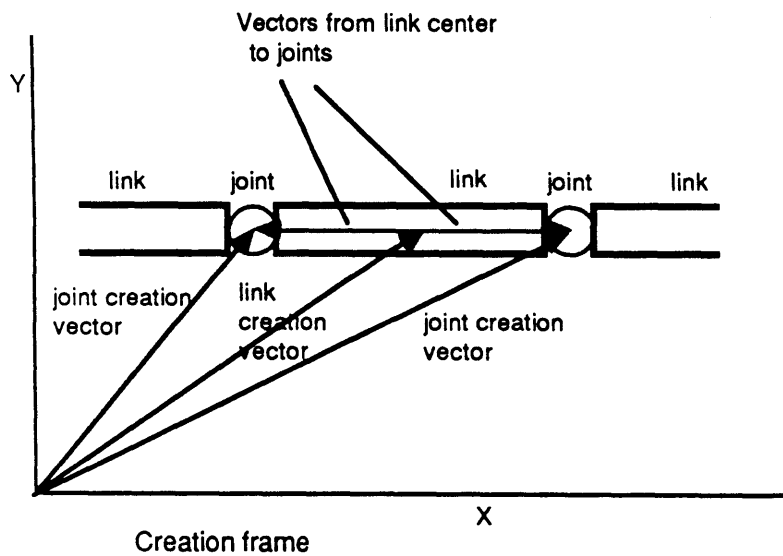


figure 5.2



From this initial joint data, other information is calculated for the joints and links. The center of mass of each link is estimated by averaging the positions of the joints attached to it. Each link calculates and stores the positions of its joints relative to its center such that the relative positions of joints to each other can be easily determined [see figure 5.3].

figure 5.3



Each joint is given a preferred angle, and a full-extension angle that will cause its links to align as straight as possible. The preferred angle defaults to zero, or the angle it was created at, but if this angle is near the full-extension angle, it is altered. Finally, each joint calculates and saves a rotation matrix and its inverse that will allow efficient calculation of rotation about its arbitrary axis. [See the appendix for more on rotation about an arbitrary axis.]

Some link is always assigned as the base link for the figure. When the base link is given a position in the world coordinate frame, each joint can calculate its world position from its parent's world position and its creation position relative to its parent's [see figure 5.4].

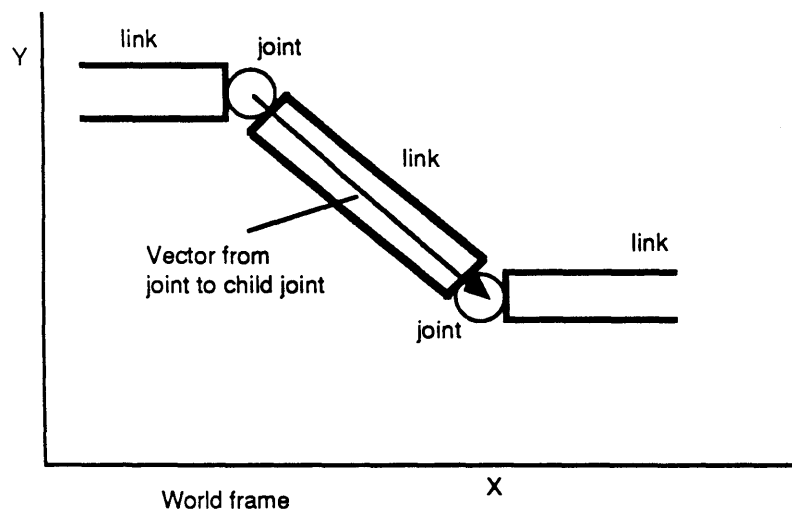


figure 5.4

An alternative form of joint representation is that of Denavit-Hartenberg (DH) notation. Four parameters, (θ, r, a, α) , for z and x axis, translation

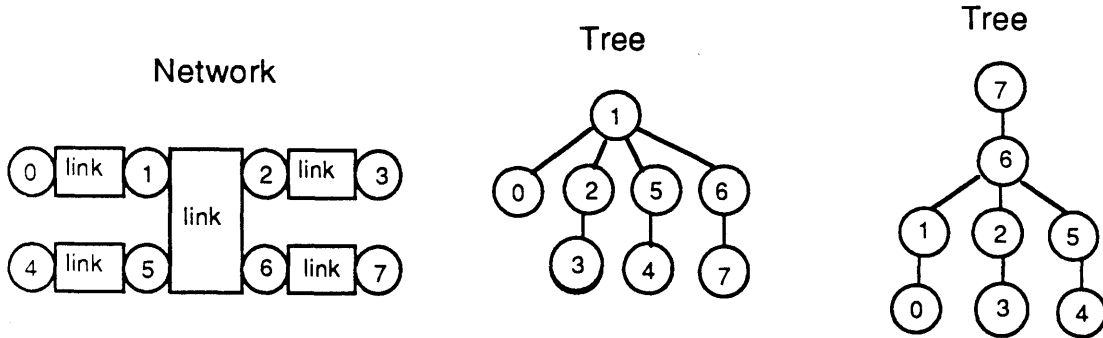
and rotation, are given for each joint in a chain that relate its position to the parent joint [44]. There are three advantages of axis and position (AP) notation over DH notation. First, it is more intuitive to specify the positions and axes of joints than it is to figure out their DH parameters, although there are now 7 numbers needed for each joint (3 for axis, 3 for position, and 1 for the current joint rotation angle) instead the 4 of DH. Second, AP representation is less directional than DH notation. More computation is needed to alter the base and tip joints of a network using DH notation. Reversing the direction of DH parameters can be confusing, but in axis and position representation the base and tip links can just be re-assigned and each joint is given a new parent joint. Third, it is easier to form branching structures. DH notation is usually used for manipulator arms consisting of a single chain of joints, there is no standard method of branching joints and links.

5.3 Joint Link Network

A jointed figure consists of a network of joints and links. There is no fixed hierarchical structure in a figure network, although a figure has a temporary tree structure. A network is used instead of a fixed tree structure so the figure can be dynamically reconfigured into new hierarchical trees when different parts of the figure act as the base [see figure 5.5]. When a new base link for the entire figure is specified, (usually the body link) each joint determines which joint if any is its new parent, which joints are its new

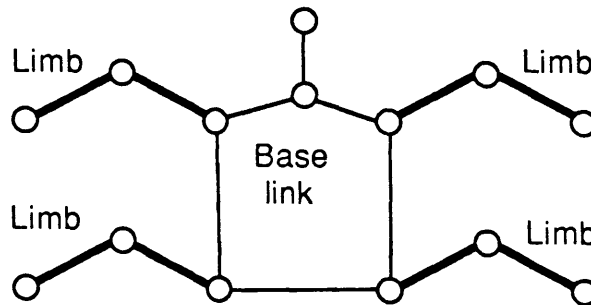
children, and what its position relative to its new parent is.

figure 5.5



Limbs are created which hold the information necessary to perform inverse kinematics between two joints: a base joint and a tip joint. A jointed figure may have any number of limbs at one time, each with a different base and tip joint. The limb bases do not have to be the same as the base of the entire figure network, although the orientation of limbs should agree with the direction of the current tree structure of the figure network [see figure 5.6].

figure 5.6, Limbs



The limbs may be dynamically rearranged. Normally, a limb represents an arm or leg where the base is the shoulder or hip, and the tip is the hand or foot, but suppose an animator wanted to position a sitting character's elbow at a particular location. A limb could be created with the base at the character's pelvis, and the tip at his elbow. Although this is not what would usually be called a limb, the animator can now specify elbow location and inverse kinematics can be used to bend the waist and shoulder to give this elbow position. A limb may also change in direction, for example, a walking inch worm has been animated where the base and tip joints are swapped between each step.

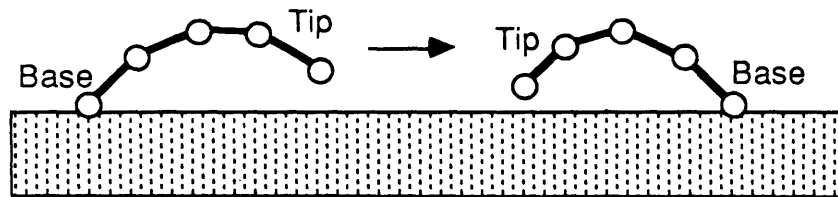


figure 5.7, Change of base and tip

5.4 Calculating the Link Positions and the Jacobian Matrix

The two primary things that need to be calculated from the jointed figure network are link position matrices and jacobian matrices. Figure 5.8

shows the subset of the modules in figure 1.1 that involve connections to the jointed figure module. Position matrices for each link must be calculated from the joint values to allow rendering of the figure. The jacobian matrix for a given base and tip joint must be calculated for use in inverse kinematics [described in the next chapter]. The necessary calculations are outlined below.

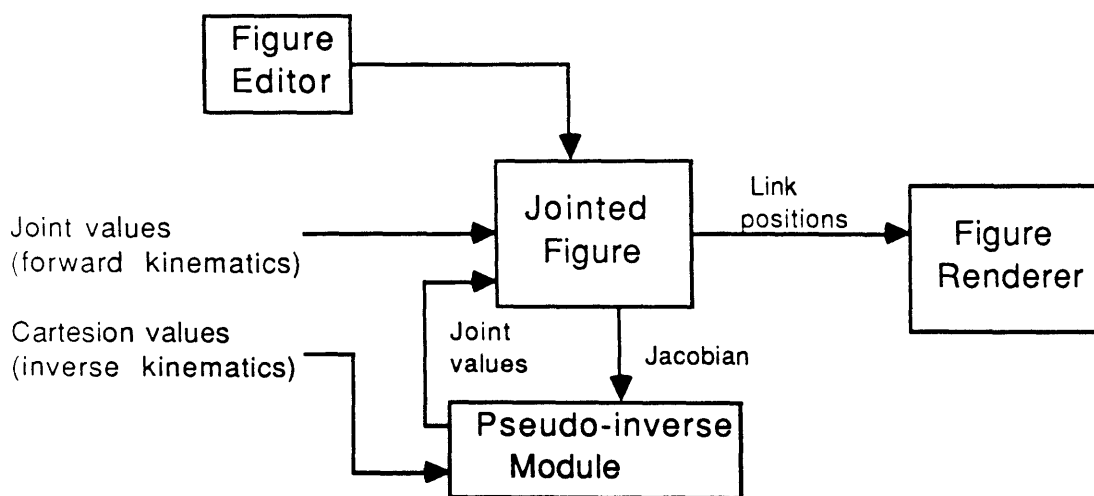


figure 5.8

The position and orientation of each joint relative to its parent is calculated by translating by its position relative to its parent, and then rotating about its axis by the amount of its current joint value.

The position matrix of each joint relative to the base is calculated by

multiplying its parent's matrix relative to the base by its position matrix relative to its parent.

The position and orientation of each link is calculated by transforming by the position of the base joint, transforming by the position of its parent joint to the base, and then translating by the distance from its parent joint.

The axes relative to the base are calculated for each joint by rotating the creation frame axis by the position matrix from the base.

A jacobian matrix is a matrix containing partial derivatives. It relates the change in each cartesian tip parameter $(x, y, z, \rho, \varphi, \psi)$ to the change in each joint angle θ_i . Here is a jacobian matrix J for a six jointed limb:

$$J = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_3} & \frac{\partial x}{\partial \theta_4} & \frac{\partial x}{\partial \theta_5} & \frac{\partial x}{\partial \theta_6} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_3} & \frac{\partial y}{\partial \theta_4} & \frac{\partial y}{\partial \theta_5} & \frac{\partial y}{\partial \theta_6} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \frac{\partial z}{\partial \theta_3} & \frac{\partial z}{\partial \theta_4} & \frac{\partial z}{\partial \theta_5} & \frac{\partial z}{\partial \theta_6} \\ \frac{\partial \rho}{\partial \theta_1} & \frac{\partial \rho}{\partial \theta_2} & \frac{\partial \rho}{\partial \theta_3} & \frac{\partial \rho}{\partial \theta_4} & \frac{\partial \rho}{\partial \theta_5} & \frac{\partial \rho}{\partial \theta_6} \\ \frac{\partial \varphi}{\partial \theta_1} & \frac{\partial \varphi}{\partial \theta_2} & \frac{\partial \varphi}{\partial \theta_3} & \frac{\partial \varphi}{\partial \theta_4} & \frac{\partial \varphi}{\partial \theta_5} & \frac{\partial \varphi}{\partial \theta_6} \\ \frac{\partial \psi}{\partial \theta_1} & \frac{\partial \psi}{\partial \theta_2} & \frac{\partial \psi}{\partial \theta_3} & \frac{\partial \psi}{\partial \theta_4} & \frac{\partial \psi}{\partial \theta_5} & \frac{\partial \psi}{\partial \theta_6} \end{bmatrix} \quad (5.1)$$

There are two types of jacobians used, the one above includes both position and orientation parameters for the tip. Another, which will be referred to as the translate jacobian J_t , only relates the tip position parameters to the joint angles.

$$J_t = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_3} & \frac{\partial x}{\partial \theta_4} & \frac{\partial x}{\partial \theta_5} & \frac{\partial x}{\partial \theta_6} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_3} & \frac{\partial y}{\partial \theta_4} & \frac{\partial y}{\partial \theta_5} & \frac{\partial y}{\partial \theta_6} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \frac{\partial z}{\partial \theta_3} & \frac{\partial z}{\partial \theta_4} & \frac{\partial z}{\partial \theta_5} & \frac{\partial z}{\partial \theta_6} \end{bmatrix} \quad (5.2)$$

The jacobian is calculated using the joint axes and joint positions relative to the base. The first three rows, are the translational partial derivatives, and are equal to the cross products of the axes with the positions of each joint from the base. The second three rows of the jacobian, are the rotational partial derivatives, and are equal to the axes of each joint. The next two chapters will contain information about how jacobian matrices are used.

In conclusion, this jointed figure representation will support several types of operations. A jointed figure can be made by giving joint positions, axes, and connectivity. Limbs can be specified with base and tip joints. Joint angles can be directly supplied (forward kinematics), or the cartesian position of a limb tip can be requested and the joint angles can be calculated using the jacobian matrix (inverse kinematics).

Chapter 6

Inverse Kinematics

When a character or a creature is manipulated for animation, the positions of the hands and feet are often what the animator is most concerned with. Usually the control of a jointed figure is performed by choosing positions or joint angles of each body part relative to the body part it is connected to. As discussed in previous chapters, it is often much more efficient if the animator can specify a hand or foot position, and the joint angles are automatically calculated by the animation system that will give this position. Inverse kinematics gives this capability [see figure 6.1].

A single unconstrained solid object has six degrees of freedom, three translational and three rotational. Most natural limbs have enough joints to give at least six degrees of freedom at their end-effector. If six degrees of freedom are specified for the end-effector (both position and orientation) but the limb has more than six degrees of freedom, the limb is called redundant or underconstrained, and there may be more than one solution to the inverse kinematics problem. If position but not orientation is specified for

the end-effector, and the limb has more than three degrees of freedom, then the problem is also underconstrained. If a limb has fewer degrees of freedom than the end-effector goal vector, then the problem is overconstrained. The pseudo-inverse provides an iterative technique for performing inverse-kinematics calculations on arbitrary limbs that may be underconstrained or overconstrained.

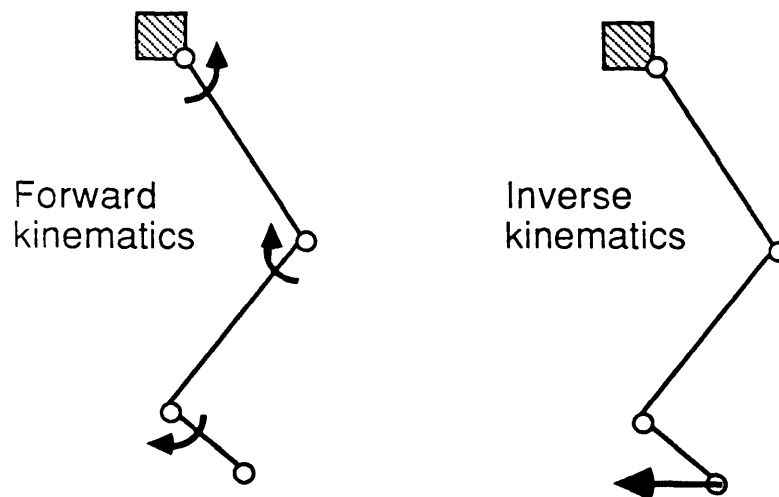


figure 6.1

6.1 Pseudo-inverse Iterations

Finding closed form solutions for the joint angles θ from a end-effector position x can be difficult because there may be many solutions as mentioned above, and because the problem is highly non-linear and intractable for all

but simple linkages.

The problem is however, linear in velocity space. It is more reasonable to relate the change of end-effector parameters \dot{x} with the change of joint values $\dot{\theta}$ and solve the inverse kinematic problem incrementally. This method is general, it can be used for any limb geometry, but the tradeoff is that more computation is needed since iteration is involved. The jacobian matrix describes how the change or instantaneous velocities of \dot{x} and $\dot{\theta}$ are related.

$$\dot{x} = J\dot{\theta} \quad (6.1)$$

The computation involved to invert the jacobian and solve this equation for $\dot{\theta}$,

$$\dot{\theta} = J^{-1}\dot{x} \quad (6.2)$$

is discussed in Appendix B.

In each pseudo-inverse iteration a desired change Δx in the end-effector is determined, and the joints are incremented by the corresponding $\Delta\theta$ from the pseudo-inverse. Since the pseudo-inverse actually relates the instantaneous velocities \dot{x} and $\dot{\theta}$, there is some error in the resulting Δx . If the desired end-effector position is a significant distance from the current end-effector position, Δx is large, and even if the method above is repeated, the limb may never converge on the desired end-effector position.

To assure the limb will reach the final desired end-effector position,

the path between the current and desired end-effector positions is divided into small segments. Pseudo-inverse calculations are performed for each segment of the path, such that Δx is kept small. For each new point on the path, a new Δx is found between the end-effector position from the last iteration, and the next point along the path [see figure 6.2].

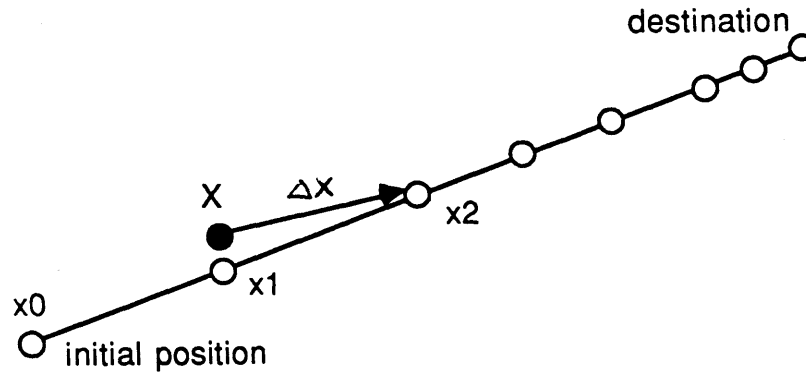
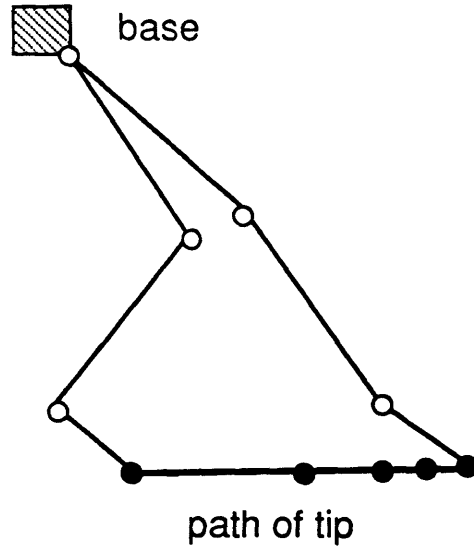


figure 6.2

The size of the segments along the path is adaptive to the magnitude of the error between the desired Δx and the actual Δx . If the error is above a certain maximum the joints are returned to their previous values, and a smaller segment is used. If the error is small, a larger segment is used [see figure 6.3]. This method has shown to be successful at keeping the pseudo-inverse calculations to a minimum, while still reaching the desired position. When a limb is traveling through an ill-behaved configuration that produces inaccurate results from the pseudo-inverse, the segment size is reduced, but when the limb is in a well behaved region, the segment size

can be large.

figure 6.3



6.2 Preferred Joint Values

Natural joints have physical constraints that cause minimum and maximum angles that they can not extend beyond. When using pseudo-inverse for inverse-kinematics, joint limits can be avoided by providing the pseudo-inverse with secondary goal information. All the joints in a jointed figure description have preferred joint values, or center angles, that are used in an attempt to keep the joint angles within their limits. If a goal is reachable and the pseudo-inverse has many solutions, the solution with the minimum difference to the preferred angles is given. If a goal is not reachable, the

pseudo-inverse will give the closest solution. [See Appendix B for details on the pseudo-inverse.]

6.3 Dealing With Singularities

There are a set of configurations that a limb can be in for which the jacobian matrix is non-invertible or singular. If the limb is near one of these configurations the Jacobian is ill-conditioned and the pseudo-inverse is subject to errors, some of it's elements may be very large, and some very small. If the limb is exactly at one of these configurations the pseudo-inverse can not be used at all. The value of the determinant of the Jacobian can be used to predict how near a singularity a limb is.

A common example of a singularity occurs when a limb is fully extended and the desired motion of the tip (or end-effector) is towards or away from the base. There is no set of joint velocities that will give a cartesian tip velocity in that direction.

There are some methods for avoiding singularities and increasing toleration of being close to singularities. The ability to deal with being near a singularity can be increased by using higher precision floating point. Singularities can be avoided by providing secondary goal information [24,26,28,29,41], as described in Appendix B.

Natural creatures seem to be able to deal with singularities just fine, and in fact often use them to their advantage. Efficient support is obtained when a leg or arm is fully extended, or even slightly over-extended. When

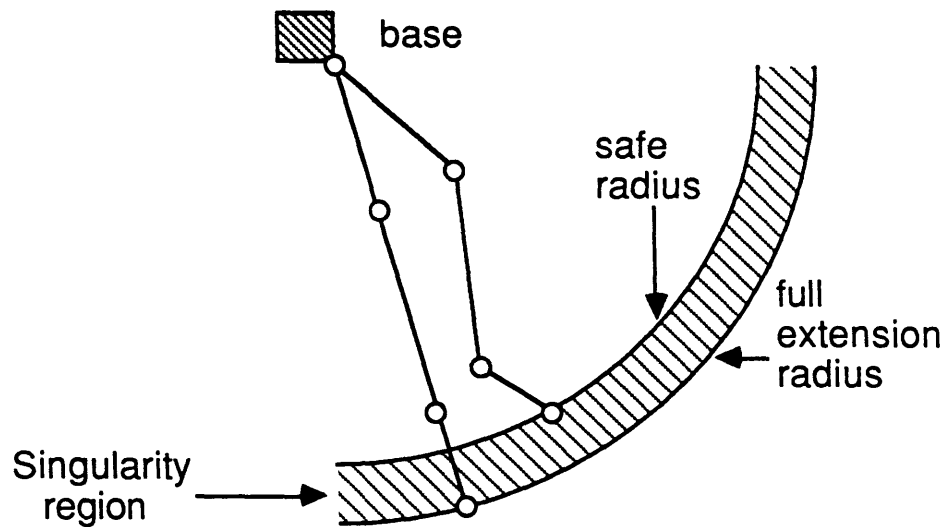
a running creature's leg first touches the ground or just leaves the ground, it is often fully extended.

To animate the locomotion of arbitrary jointed figures, some method for allowing limbs to fully extend seemed necessary. Using the pseudo-inverse and avoiding singularities, or just increasing tolerance of being close to them, was not sufficient.

6.3.1 Allowing Full Extension

First a method was developed that can detect when a limb is within a region near the singularity of full extension. If the distance between the base and tip of a limb is near the fully extended length of the limb, the limb is in this dangerous singularity region.

figure 6.4



If a limb is moving from the singularity region or into the singularity region, a different method for manipulating the limb is used. This method assumes that a limb contains a ball joint at its base. The tip position is reached by first finding joint angles that put the tip at the correct distance from the base, and then the base ball joint is adjusted to position the tip exactly at the requested position.

Each joint has a preferred value that causes it to be bent one way or another, not straight out. If all the joints of a limb are at their preferred values, the limb should be out of the singularity region shown in figure 6.4. Each joint also has a full-extension value that causes it to be as straight out as possible.

When the limb tip is moving through the singularity region, the joint angles that give the desired distance from the base are found by performing a binary search between two sets of joint values: if the limb is getting shorter: from the current configuration, towards the preferred joint values, if the limb is getting longer: from the current configuration, towards the full extension joint values. The desired distance should be between the distances of these two limb configurations.

Once the joint angles have been found that place the tip at the requested distance from the base, the ball joint at the base is adjusted. Pseudo-inverse kinematics for the two degrees of freedom of the ball joint is used to position the tip at the desired location.

The limitations of this method are:

1. The limb must have a ball joint at its base.
2. It only deals with singularities at the full extension configuration.
3. The orientation of the tip can not be specified, only the location.

However, this has been shown to be effective in performing inverse kinematics when the limb is near or at full extension. It has been sufficient to allow the inverse kinematics necessary for locomotion.

Chapter 7

Dynamics

It is difficult for kinematically defined motion to look dynamically correct [13,14,2,72,68,6]. Forces are what actually cause physical things to move. It is physically unrealistic to specify just a series of positions to generate motion, unless these positions are carefully calculated with the laws of physics in consideration.

Unfortunately, it can be difficult to create complex motion such as locomotion by specifying only forces and torques (from muscles), and simulating the dynamics of the system to get the resulting motions. This is difficult for two reasons:

1. It may be difficult to calculate a complete dynamic simulation for a complex system of connected joints and links [72,2].
2. It is difficult to calculate the necessary forces and torques to give a desired complex motion.

Some areas of robotics research are faced with the problem of item 2

above [10,16,48,49,51,27,68]. Fortunately for computer graphics, computations do not need to be performed in real time as they do in robotics, and motion doesn't necessarily need to be correct as long as it looks correct. It is important, however, to realize how sharp humans are at perceiving and predicting dynamics.

The extreme of complete force generation and dynamic simulation is the ultimate way of achieving dynamically correct motions, and may well eventually be the best way to create realistic movements for animation. In this work, simplifications have been made that allow easier motion calculation, but some dynamic simulation is performed and some amount of dynamic correctness is still achieved.

The following sections of this chapter discuss what is involved in dynamic simulations of single bodies and of articulated figures, some possible methods of finding muscle forces are mentioned, and finally hybrid dynamic-kinematic control is discussed.

7.1 Dynamic Simulation of Single Rigid Bodies

Given a rigid solid body with forces acting on it, its motion can be simulated with reasonable dynamic accuracy using the following model. The translational and rotational effects of the forces are found independently, and the translational and rotational variables are calculated separately to

incrementally give new positions and orientations of the object.

Time is divided into small increments of duration Δt that are shorter to the time of a single frame of animation, or 1/30 second. For each time increment, the acceleration is found, velocity is updated from acceleration, and finally position is updated from velocity. This is done for both translation and rotation.

A body is represented by a set of n point masses m_i that sum to a total mass m_T . The state variables of an object are position P , an orientation matrix R , linear velocity V_t , angular velocity V_θ , and forces F_i acting on the body at specified locations L_i . Gravity is usually one of the forces acting on the body at its center of mass.

The translational acceleration A_t is simply the sum of the force vectors divided by the total mass:

$$A_t = \frac{\sum F_i}{m_T} \quad (7.1)$$

and the translational velocity is

$$V'_t = V_t + \int A_t dt \quad (7.2)$$

If the simulation time increment Δt is small enough, the quantity being integrated can be approximated to be constant, so the velocity above can be rewritten as

$$V'_t = V_t + A_t \Delta t \quad (7.3)$$

The new position of the center of mass is

$$P' = P + \int V_t dt \quad (7.4)$$

again, this can be approximated by

$$P' = P + V_t \Delta t \quad (7.5)$$

Calculating the rotational effects of the forces on the body is somewhat more difficult than the translational effects, but the same strategy is used. The torques are calculated using the direction and magnitude of the force vectors F_i and also the locations that the forces are acting at. If the moment arms L_i are the locations of the forces relative to the center of mass of the body, the torque is $F_i \times L_i$, and the total torque T is given by:

$$T = \sum F_i \times L_i \quad (7.6)$$

The moment of inertia is a function of the axis \hat{T} that the torque is exerted about. The angular acceleration of the body is the total torque divided by the moment of inertia:

$$A_\theta = \frac{T}{i(\hat{T})} \quad (7.7)$$

and the new angular velocity is

$$V'_\theta = V_\theta + A_\theta \Delta t \quad (7.8)$$

The new orientation of the body is the previous orientation rotated about the axis of V_θ by an angle $|V_\theta|\Delta t$. If the orientation of the body is described by a matrix R then

$$R' = R \text{Rot}(V_\theta \Delta t) \quad (7.9)$$

where $\text{Rot}(V_\theta \Delta t)$ is a transformation matrix that rotates by an angle of $|V_\theta \Delta t|$ about the axis V_θ .

7.1.1 Calculating Moment of Inertia

A sphere of uniform mass has the same moment of inertia for any axis of rotation. Objects that are nearly spherical can have their moment of inertia approximated to be a constant, independent of the axis of rotation, with reasonable results.

Objects that are not very spherical, need to have their moment of inertia calculated for each new axis of rotation. For example, it takes significantly more torque to reach the same angular velocity when a pencil is rotated from end to end then when it is spun lengthwise. If the mass distribution of the object is approximated by a set of point masses, the moment of inertia can be calculated for any axis of rotation. For a set of n point masses of mass m_i at relative position L_i from the center of mass of the object, the moment of inertia i about the axis unit vector \hat{T} is given by:

$$i = \sum_{i=0}^n m_i (\hat{T} \times L_i)^2 \quad (7.10)$$

A moment of inertia matrix could be calculated by finding the moment of inertia as described above for the 3 primary axes. A matrix created from these results could give the moment of inertia about any axis by multiplying this matrix with the axis vector. In the current implementation this method is not used. Instead the equation above is solved for each new axis.

7.2 Dynamic Simulation of Articulated Figures

When rigid bodies are connected together into a network of joints and links, the dynamic simulation of the system becomes much more complex than that of a single body described above.

The dynamic equations for each link can be expressed which include forces and torques from connected links, from centripetal forces, coriolis forces, and external forces. The equations from each link can then be combined and solved for the acceleration of each joint [2,72,68]

There are several methods for solving the dynamic equations for the joint accelerations [68,2]. Most methods are order N^3 where N is the number of links. An order N^2 method has been developed that can pay off for a large number of links, but it is slower for $N = 6$ [68]. A technique has been described that approximates the dynamics for animation of articulated rigid bodies in linear time [2].

The details of solving the dynamic equations are not discussed here

since they were not used, refer to [68] and [2] for comparisons of methods of solving dynamic equations and the use of dynamic simulation of articulated figures for computer animation.

7.3 Generating Muscle Forces

Given a system that could simulate the dynamics of a system of joints and links from a set of joint forces or torques, the problem is to generate these joint torques that will produce a desired motion. This is a common but difficult problem in robotic locomotion [10,16,48,49,51,27].

7.3.1 Inverse Dynamics

If a trajectory of a tip of a limb is known, the necessary joint forces to create that trajectory can be found using inverse dynamics. This is a useful technique in some situations such as robot control, but for computer graphics animation it is usually not necessary, because the original trajectory itself could more easily be used to position the limb tip with inverse kinematics. However, it may be useful in calculating the forces and the amount of energy required to produce a given trajectory.

7.3.2 Minimum Energy

In natural locomotion, it makes sense that movements would consume the smallest possible amount of energy but still perform the desired result of self transportation [1,3,8,19,20,22,36]. The energy used in natural locomotion is roughly the sum of the energy used by each muscle. The energy used

during natural locomotion is often measured by monitoring total oxygen consumption. The energy used by muscles can be approximated by the work done by them which equals the exerted force times time.

One could imagine a system that could be given a trajectory with some constraints, and vary the trajectory within these constraints, until a minimum energy trajectory is found. For example, a stepping foot should move from one point on the ground to another point on the ground without touching the ground in between. There are many trajectories that could accomplish this, but the one of minimum energy is probably a fairly realistic choice. The speed and path of the trajectory could be adjusted towards a trajectory of lower energy, until a minimum is reached. The total energy needed to produce this trajectory would be measured by the integral of the joint forces over time. This would approximate the biological energy needed to exert muscle forces to cause this trajectory.

When computing muscle forces, maximum allowable forces should also be considered. In natural muscles there are physical maximum forces that can be exerted, these limits should be obeyed to avoid motions that might look unnaturally jerky.

7.4 Hybrid Dynamic-Kinematic Control

Dynamic control and kinematic control can be combined to avoid some of the complex problems discussed above but still produce dynamically correct looking motions. A jointed figure can be divided into sections, and each

section controlled independently. Some groups of links can be controlled dynamically, and some kinematically.

For example, in the work described in this thesis the legs of a figure can be controlled kinematically, and the body can be controlled dynamically. Given a trajectory of the body and trajectories of each foot, the legs can be positioned with inverse kinematics, but the motion of the body of the figure can be generated by a dynamic simulation with forces from the legs and gravity acting on the body of the figure.

For some types of locomotion such as hopping, this method works well, but for other types such as running, it can be difficult to determine what forces from the legs will give a desired body motion.

Another example of hybrid dynamic-kinematic control, although this one has not been implemented, would be to solve the dynamics for a single swinging leg given a body path. When a leg first swings forward, the lowest energy trajectory occurs when no joint forces are exerted. Each leg could swing forward freely and at some point inverse kinematics could take over to position the foot at its final destination. It would hopefully not be necessary to consider the forces acting on the body from the swinging leg.

Hybrid dynamic-kinematic control has been used to generate the motions of hopping creatures. The motion of the body is determined using dynamic simulation, and the motion of the legs is determined kinematically. Dynamic control has not been used in the generation of the motions for walking creatures.

Chapter 8

Terrain

For legged locomotion to occur, there must be some kind of terrain represented such that the feet can be placed on it and support the figure. The simplest type of terrain to represent, and the easiest to create locomotion over, is a completely flat surface. Most animation of locomotion has been done using flat terrain [13,14,12].

In the work described in this thesis, the terrain can be a complex surface, and motions for locomotion are generated assuming arbitrary terrain shape [see figure 8.3]. Terrain is used to create an environment that can be both rendered, and sensed by the characters in it. Feet must be able to detect when they have made contact with the surface of the terrain, so the height of the terrain at any location must be calculated quickly.

8.1 Fast Surface Height Detection

The following method has been used to allow fast surface height calculations for bumpy terrains. Three dimensional objects used for terrain are

constructed with regular arrays of polygons, either triangles or quadrilaterals. When the surface is initially created, the polygons of the surface are stored in such a way that any horizontal world space coordinates, (x_w, z_w) , can be quickly converted into polygon space coordinates, (x_p, y_p) , by a simple transformation. If the surface object is transformed to a new position, the inverse of this transformation is used to find the polygon coordinates. The polygon coordinates determine a single polygon that is below (x_w, z_w) , the heights at the vertices of this polygon are linearly interpolated to give the exact polygon height at (x_p, y_p) [see figure 8.1]. Then this height is transformed back to world space by the surface object's position matrix to give the height, y_w , of the surface in world coordinates.

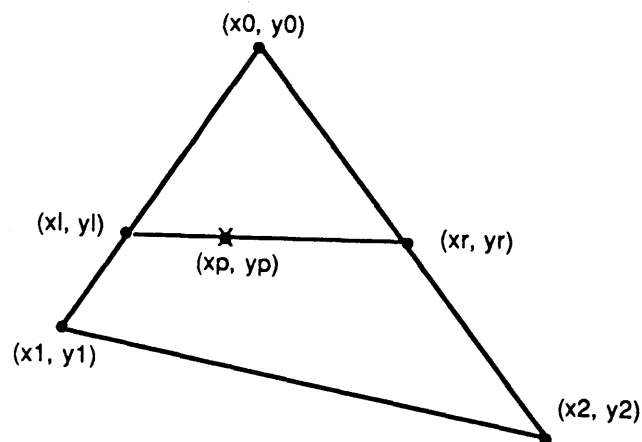


figure 8.1, polygon interpolation

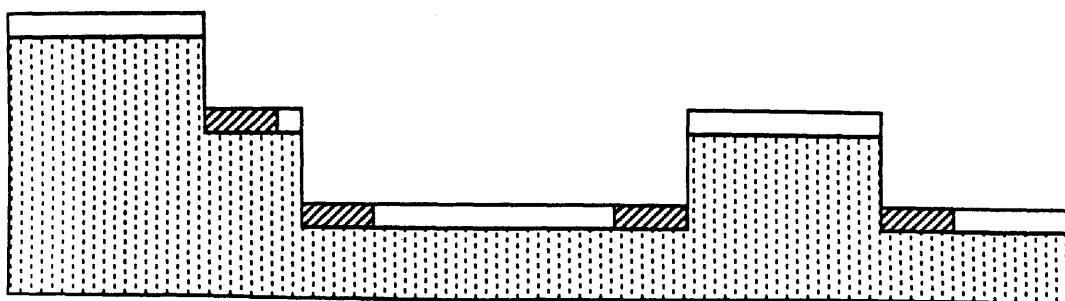
8.2 Forbidden Zone Detection

One of the many skills involved in natural locomotion, is the detection of bad footholds. Consider the perception involved in walking quickly over a rocky beach. Humans can distinguish between acceptable and unacceptable foothold locations at incredible speeds, just by looking at the terrain [69].

Fortunately, computer graphics doesn't need to use vision techniques to find acceptable footholds. The representation of the terrain can be directly used to allow distinction between good and bad foothold locations.

Given an unstepable slope value, and a foot-radius, the forbidden zones can be defined. Any area on the terrain within one foot-radius down hill of an unstepable slope will not make an acceptable foothold [see figure 8.2]

figure 8.2, Bad Foothold Locations






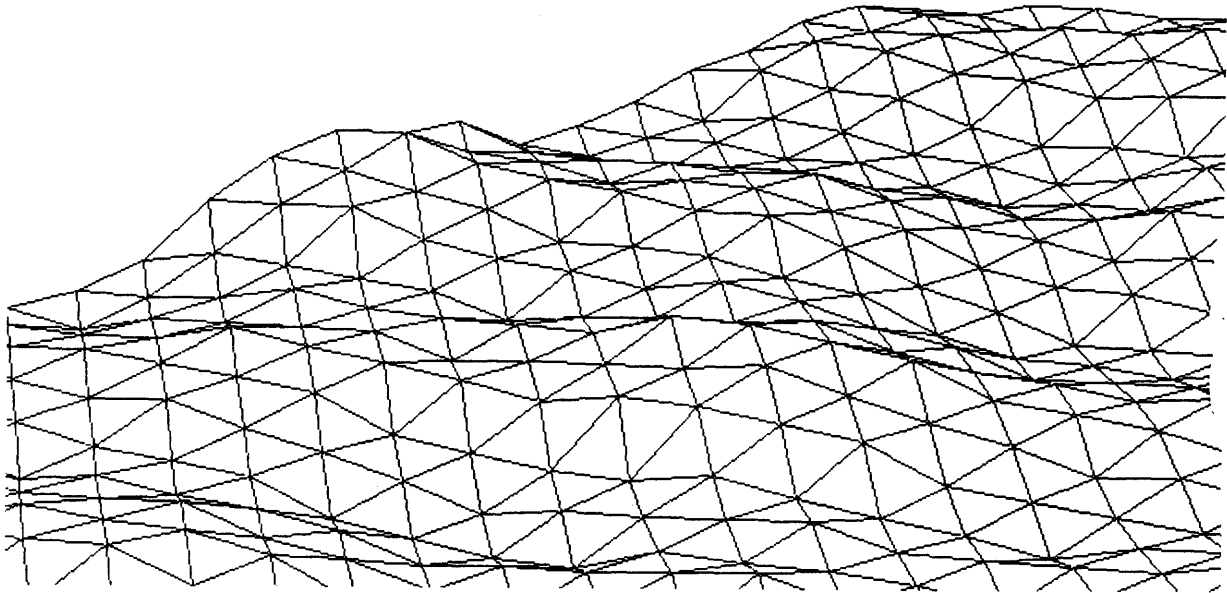
Forbidden zones 
Safe zones 
Foot radius 

figure 8.3, Example of Terrain Used



Chapter 9

Locomotion

There is an incredibly large variation in methods that creatures use to transport themselves. They can be categorized to some degree but each category still has its own significant variations. Flying, swimming, walking, running, hopping, crawling, galloping, and trotting, are some examples of names for categories. Walking alone has many variations that are important to be able to create in a complete animation system. The style of walking can express a lot about a characters personality or mood.

9.1 Locomotion Parameters

Different styles of locomotion must be represented in a way that an animation system can interpret and attempt to generate motions for them. There is no obvious set of locomotion parameters that can describe all styles of locomotion. Given a large set of parameters, some may be necessary to specify sometimes and others at other times, and many parameters should often be calculated automatically or defaulted.

For example, the locomotion parameters for walking might be:

1. Gait
2. Stride cycle time
3. Speed
4. Bounciness
5. Body height above the terrain

Some different parameters might apply for a hopping (or pronking) style of locomotion:

1. Jump distance
2. Jump height
3. Crouch height
4. Speed

There may be constraints between some of these locomotion parameters because they are not independent. In the set of locomotion parameters given above for hopping, jump height, jump distance, and total speed are dependent on each other. Speed and gait are often related, some gaits look funny at certain speeds.

9.2 Gaits

Representations for gait patterns of legged creatures have been proposed for many years [40,19]. There is a tradeoff in representing gaits between the amount of specific gait information contained in the representation and the ease which a gait can be described and represented visually. Muybridge [40] used a diagram that described only the order of foot placement:

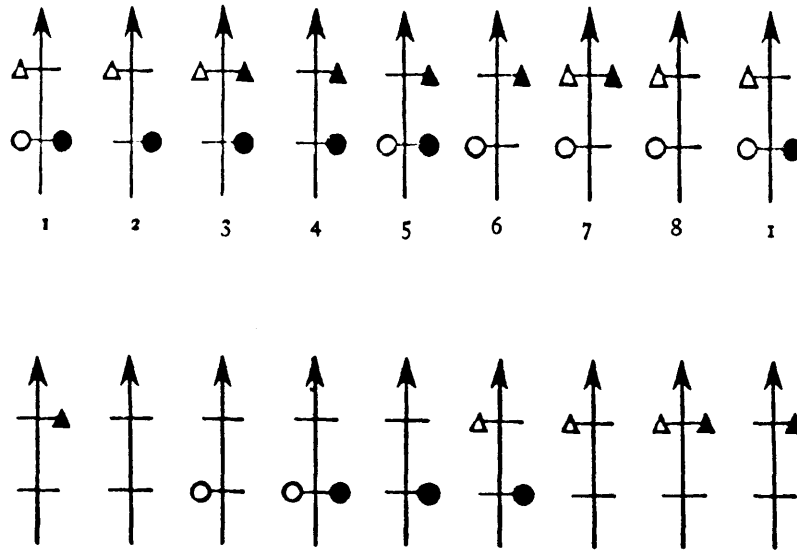
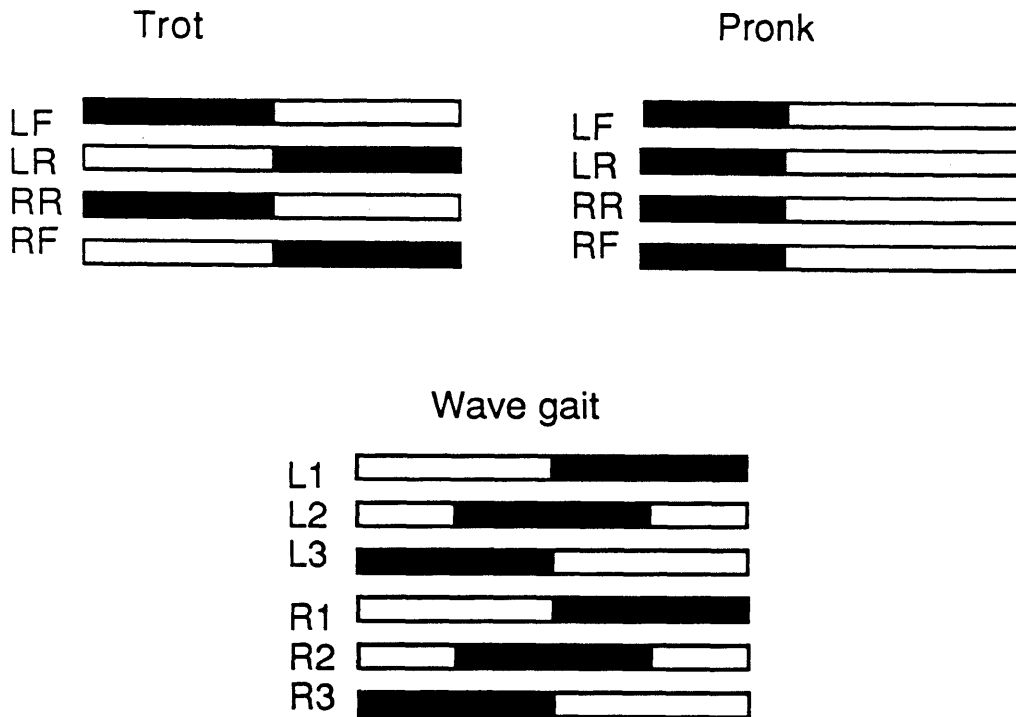


figure 9.1, walking and galloping

Other representations give the relative timings for placing and lifting of each foot. In figure 9.2, the horizontal dimension represents time, black indicates when the foot is in contact with the ground.

figure 9.2



In this work, it was important for the gait representation to be specific enough that exactly timed motions of the legs and feet could be generated. The parameters used to describe a gait are the times that each foot is lifted up and placed down, and the total gait cycle duration time. The foot-up and foot-down times are stored as times relative to the gait cycle (from 0.0 to 1.0). The phase of the gait cycle is increased as the jointed figure walks along.

Using this representation the gait can easily be sped up or slowed down

by only changing the cycle time parameter while the gait pattern remains unaffected. The speed of a gait may want to be adaptive to the terrain, a creature might automatically slow down and take shorter steps when climbing up hills, or speed up and take longer steps when traveling down hills.

9.3 Planning Trajectories

When creating motion for locomotion, trajectories for both the body of the figure, and for each foot of the figure must be generated. A trajectory can be planned or not planned. A planned trajectory might be used when the destination of the trajectory has some constraints. If a hopping creature wants to avoid tripping over obstacles, it should plan a trajectory for its body that will be clear of any. If a walking creature wants to avoid stepping into ditches or stubbing the terrain with its feet, it should plan the trajectories of its feet so they avoid protrusions and have safe destinations. The next chapter contains some details on planning foot trajectories.

9.4 Not Planning Trajectories

There are many circumstances where it may be reasonable to not plan trajectories. Forces can be applied and the trajectories can be calculated incrementally with dynamic simulation. Besides the fact that dynamically correct trajectories can often be difficult to predict and plan, natural locomotion probably does not always involve careful planning. When we walk

or run, we get used to the “rhythm” of it, and don’t have to carefully place each foot. Real creatures and humans sometimes don’t plan trajectories as well as they should and they can trip and fall.

Unplanned trajectories can still be somewhat adaptive to complex terrain. For example, a creature may jump into the air not knowing where it will land, but when the creature’s legs first touch the ground, they exert the necessary forces to cause a gentle landing on the surface.

If our creature is just learning to walk, or if a creature encounters an unusual change in height of the terrain, it may fail to stay upright. With unplanned trajectories this can actually happen, creatures have been animated that have tripped and fallen over when a severe bump in the terrain was encountered.

Chapter 10

Walking

A figure transporting itself in such a way that at least one foot is always on the ground will be considered to be walking. Since the body is always supported by at least one leg when walking, it is never in free flight. Walking is probably the easiest style of locomotion to generate motion for, since the movements of both the body and the legs can be approximated with kinematic techniques that do not involve extensive dynamic considerations.

The problem of generating walking motion is divided into two parts: calculating the body trajectory, and calculating trajectories for the feet. Once the body and feet trajectories have been chosen, the leg joint angles that give the desired foot positions from the body location can be easily determined using inverse kinematics.

The timing of the trajectories of the feet is determined by the gait information and the phase of the gait cycle time. The body path can also be affected by the gait cycle phase.

10.1 Body Path

There are several options for types of body paths for walking figures. The body can move along a simple straight line, this creates what is known as the refrigerator look [13]. A linear body path is also not sufficient for walking over complex terrain, when a figure reaches a small hill, the body would just ram into it.

A body path can be adaptive to the terrain. It can translate up or down depending on the terrain height, or it can pitch forward and back depending on the slope of the terrain. The slope of the terrain could also be used to adjust the speed of the gait. This allows figures to walk over uneven surfaces, but the body motion still often looks too smooth or mechanistic.

A body path can be affected by the gait cycle phase. The body should bounce up and down slightly as the figure exerts forces with its feet. A cyclic spline curve can be used whose phase depends on the gait cycle time and amplitude is the bounciness of the figure. When the effects from the gait cycle phase and the terrain height and slope are combined to determine a body path, a lively looking walking motion can be produced that is adaptive to the terrain.

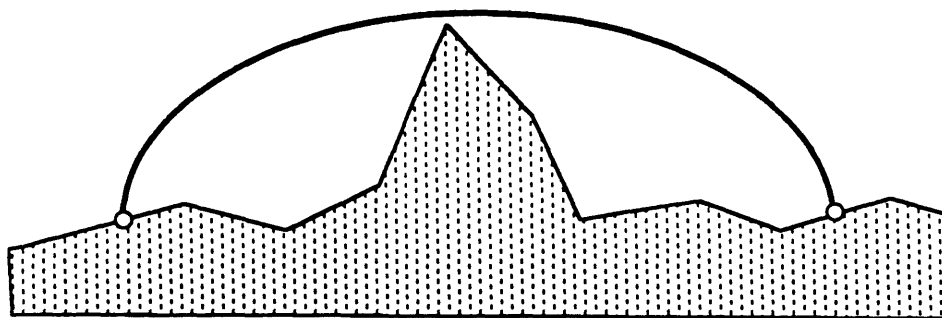
10.2 Feet Trajectories: Step Planning

The gait of a walking figure determines when each foot is lifted up and placed down. When the gait cycle phase reaches a foot-up time, a new

trajectory that will bring it to its next foothold location must be planned automatically for that foot. A foothold position for the next contact is found, and a path leading to that foothold that will avoid collisions with any protrusions in the terrain is determined. The flight of the foot along this path is timed so that the foot lands according to the foot down time of the gait. Here is the process for planning a step trajectory from one foothold position to the next:

1. Find the foothold location at the desired distance and direction.
2. If that is a bad foothold location, find the farthest good foothold location that is closer than that.
3. Find the maximum terrain height between the current foothold and the new foothold.
4. Create a trajectory with a spline curve that clears that height.

figure 10.1, path over protruding obstacle



Once the body path and feet trajectories have been determined for a walking figure, the body can be positioned at the proper location for each frame of the animation, and the feet are positioned at the position in their world coordinate trajectories using the inverse kinematics techniques described in chapters 6 and 7.

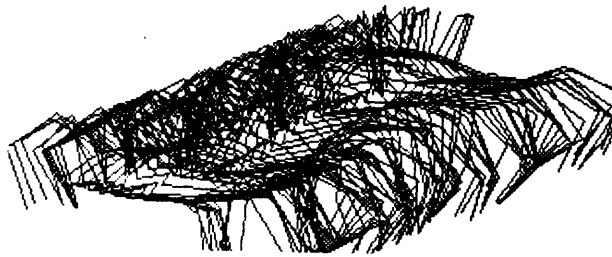
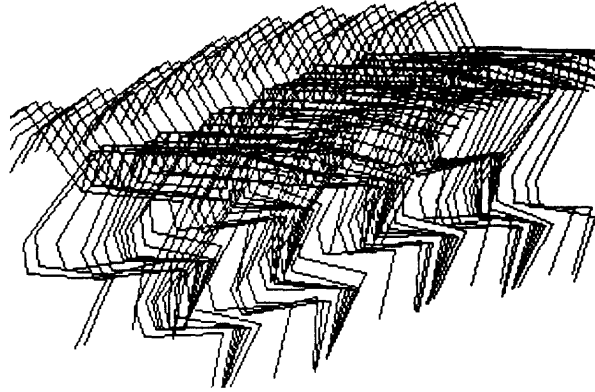
10.3 Automatic Gait Assignment

Two generalized types of gaits can be automatically assigned to a jointed figure with any number of legs: a “trot” or a “wave gait”. A “trot” will give the standard trot for quadrupeds, a tripod gait for hexapods, or a normal walk for bipeds. A “wave gait” will give the usual wave gait in creatures with greater than four legs, a bound for quadrupeds, or a hopping gait for bipeds.

The legs of a jointed figure are separated into left and right sides, and then sorted from front to back. The foot-up and foot-down times are determined for the left and right side independently, so the wave gait will be adaptive to missing limbs.

In figure 10.2 the frames from a sequence of animation have been composited in an attempt to show the walking motions that can be generated.

figure 10.2, examples of walking



Chapter 11

Pronking

Pronking is a type of locomotion where all the feet contact the ground simultaneously, and leave the ground simultaneously. Pronking could also be called hopping or jumping. Pronking is common in some bipeds such as kangaroos, and is sometimes performed by quadrupeds such as antelopes or gazelles [19].

Pronking has been simulated for arbitrary jointed figures that have been created with the figure editor described previously. The gait cycle for pronking has only two states: flight phase and support phase.

Since pronking is simulated on uneven terrain, the landing time of all the feet is usually not quite simultaneous. A gait will be considered to be in support phase only when all the feet have made contact with the ground. For the purposes of simulation, the support phase will be divided into two phases, the landing phase and the pushing phase, so the entire gait cycle will have three phases:

1. Flight phase

2. Landing phase

3. Pushing phase

The path of the pronker's body is determined using dynamic simulation. During flight phase, gravity causes a constant acceleration of the pronker toward the ground causing a smooth parabola. During landing phase and pushing phase the legs exert forces on the body at the hips that cause the pronker to decelerate and push off for the next jump. For details of the dynamic simulation techniques used, refer to the section about dynamic simulation of single bodies in the chapter on dynamics.

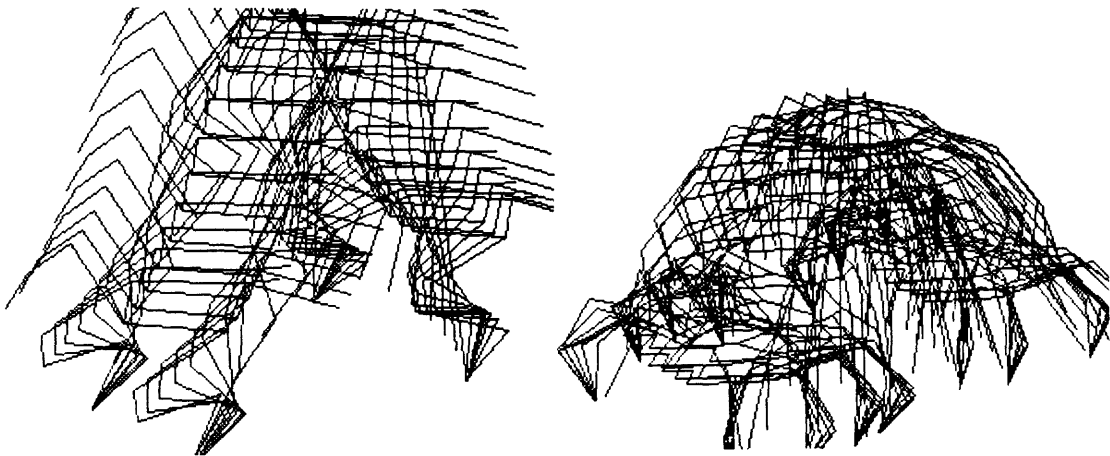


figure 11.1, Pronking Biped and Pronking Quadruped

Inverse kinematics is used to determine the leg joint angles that position the feet at given foothold and swing trajectory locations when the body position is known.

The transitions between the three phases are determined by specific events that allow pronking to be adaptive to uneven terrain. The transition between flight phase and landing phase occurs when all the feet of the pronker contact the ground. The transition between landing phase and pushing phase occurs when the pronker reaches a vertical velocity of zero. And the transition between pushing phase and flight phase occurs when any of the pronkers feet leave the ground, or when the pronker reaches the desired jumping velocity.

11.1 Flight Phase

During the the flight phase, as each foot can not reach the ground anymore, it is given a trajectory relative to the pronkers body that will swing it forward for the next landing. The total flight time is estimated from the current vertical velocity v_{y0} . If we assume the final velocity v_{y1} will be equal but negative of the current velocity, the total flight time Δt_f can be found as follows:

$$v_{y1} = -v_{y0} \quad (11.1)$$

$$v_{y1} = v_{y0} + g\Delta t_f \quad (11.2)$$

$$\Delta t_f = \frac{2v_{y0}}{g} \quad (11.3)$$

This Δt_f is used to determine the duration of the trajectories for the feet from their current positions to their final positions relative to the body. These paths are specified with three spline points, the center one is slightly

higher than the two endpoints so the leg moves toward a relaxed position when the pronker is ascending, and extends forward for the next landing when the pronker is descending.

During each simulation iteration of the flight phase, the feet are tested for intersections with the terrain. If any foot does intersect the terrain, the point on the surface of the terrain is stored and becomes the foothold location for that foot during the landing and pushing phases.

11.2 Landing Phase

The landing phase is initiated when all the feet of the pronker have made contact with the ground. During the landing phase each leg exerts a constant force on the body to absorb the vertical momentum. The crouch height y_c is the locomotion parameter of a pronker that describes the preferred height above the terrain that the pronker should reach zero vertical velocity at.

A strategy used to find the appropriate leg forces in both the landing and pushing phases will be to first determine a desired change in velocity, ΔV , and a desired change in height Δy . From the vertical change in velocity Δv_y and Δy the necessary time duration, Δt , can be found. Then the accelerations and forces that will give the desired change of each component of ΔV in that time duration can be found.

For the landing phase the desired vertical velocity is zero, so if the initial landing vertical velocity is v_{y0} , then

$$\Delta v_y = -v_{y0} \quad (11.4)$$

If the initial landing height is y_0 and the desired crouch height is y_c then

$$\Delta y = y_0 - y_c \quad (11.5)$$

From these we can find the necessary Δt :

$$\Delta y = \int v_y dt \quad (11.6)$$

since the acceleration will be constant, velocity will change linearly and

$$\Delta y = \frac{v_{y0} + v_{y1}}{2} \Delta t \quad (11.7)$$

$$\Delta t = \frac{2\Delta y}{v_0 + v_1} \quad (11.8)$$

Next the force vector F that will give the desired ΔV in this Δt will be found. The change in velocity is related to the acceleration, A , by

$$\Delta V = \int A dt \quad (11.9)$$

Again, since the acceleration will be constant

$$\Delta V = A\Delta t \quad (11.10)$$

$$A = \frac{\Delta V}{\Delta t} \quad (11.11)$$

Finally, to find the forces that will give these accelerations we also need to compensate for the mass of the pronker m , and the effect of gravity:

$$F = m(A - G) \quad (11.12)$$

where G is the gravity acceleration vector.

This total force is split up equally amongst the legs of the pronker. If each of n legs of a pronker exerts a force of F/n during the landing phase, then the pronker should reach the desired crouch height with a zero vertical velocity. If the center of mass of the pronker is also the centroid of the hip positions, then these forces will not produce any torque and the pronker should remain level. Each leg has a maximum force that it can not exert more than. If the desired force exceeds the maximum the leg will exert the maximum instead, and the pronker will crouch below the desired crouch height and might even hit the ground. This will happen if a pronker jumps off a sharp incline or is dropped from a height beyond the normal jumping height.

11.3 Pushing Phase

The pushing phase is initiated when the vertical velocity reaches zero during the landing phase. The pushing phase is similar to the landing phase in that a constant force is exerted by the legs to cause a desired change in velocity within a certain change in height. In the pushing phase the final velocity V_1 will determine the trajectory for the following flight phase, it will determine where the pronkers next landing location will be and how high the pronker will jump. This desired final velocity is calculated from

the desired jump height and jump distance of the pronker.

The estimated time duration between liftoff and the maximum height, $\Delta t_{f/2}$ can be found using the desired jump height, Δy_j as follows:

$$\Delta y_j = \frac{1}{2}g\Delta t_{f/2}^2 \quad (11.13)$$

$$\Delta t_{f/2} = \sqrt{\frac{2\Delta y_j}{g}} \quad (11.14)$$

If j_x and j_z are the desired horizontal jump distances, then the velocity that will produce the specified jump is

$$V_1 = \begin{bmatrix} \frac{j_x}{2\Delta t_{f/2}} \\ -g\Delta t \\ \frac{j_z}{2\Delta t_{f/2}} \end{bmatrix} \quad (11.15)$$

The desired jump distances could be calculated by specifying the next landing location. The jump height could also be calculated by specifying the jump distance and the total pronker speed.

If the current pronker velocity is V_0 then the change in velocity that will produce the desired jump is

$$\Delta V = V_1 - V_0 \quad (11.16)$$

If the estimated maximum height that all legs will still reach the ground is y_1 and the current height is y_0 then

$$\Delta y = y_1 - y_0 \quad (11.17)$$

From here, the same equations given above for the landing phase can be used to give a constant force vector F that can then be divided up amongst the pronkers legs to give the desired liftoff velocity. When this velocity is reached or when any of the pronkers legs can not reach the ground anymore, the flight phase begins and the cycle repeats.

Chapter 12

Bouncing

Consider the cycle of a bouncing object. As an object above a surface falls, its potential energy is transformed into kinetic energy. When the object hits the surface, the materials of the surface and the object deform and the kinetic energy is transformed into potential energy due to the springiness of the materials. Then the materials spring back and push the object back up into the air as kinetic energy again. If the surface is assumed to have infinite mass, its velocity and therefore its kinetic energy will remain at zero. If no energy is lost, the collision is called elastic, and the object would bounce back to its original height. If the object loses energy on each bounce the collision is inelastic.

12.1 Bouncing in One Dimension

First a model will be proposed for an object bouncing up and down in one dimension. We can describe the elasticity of an object as the percent of kinetic energy lost, or the ratio of the kinetic energy before and after a

bounce. An object with mass m has a kinetic energy:

$$K_e = \frac{1}{2}mv^2 \quad (12.1)$$

If the object has an initial velocity v_1 and a final velocity v_2 the kinetic energy ratio E_r is:

$$E_r = \frac{\frac{1}{2}mv_1^2}{\frac{1}{2}mv_2^2} = \left(\frac{v_1}{v_2}\right)^2 \quad (12.2)$$

For a rubber ball this might be 0.8 or for a baseball around 0.2. This kinetic energy ratio also equals the ratio between successive heights the objects bounces. The kinetic energy at the point of zero height equals the potential energy at the point of zero velocity and maximum height, and potential energy E_p is proportional to height h :

$$E_p = mgh \quad (12.3)$$

So if there is no air friction:

$$E_r = \left(\frac{v_1}{v_2}\right)^2 = \frac{h_1}{h_2} \quad (12.4)$$

Notice that the kinetic energy ratio is simply the square of the momentum ratio, or the velocity ratio. I will use this momentum ratio, call it b , to describe the bounciness in calculations just because it is slightly simpler than the energy ratio.

$$b = \frac{m|v_1|}{m|v_2|} = \left|\frac{v_1}{v_2}\right| = \sqrt{E_r} \quad (12.5)$$

12.2 Simulating Bounces with Surface Forces

When a collision of an object with a surface is detected, forces can be applied to that object for a certain amount of time Δt that cause it to bounce away from the surface. If the simulated object and surface are rigid or non-deformable, a bounce should happen instantaneously as soon as the object contacts the surface, and Δt should equal the time duration of a single simulation iteration. If the object or the surface is deformable, then the bounce should have more duration, Δt should be larger, and the force to give the bounce will be applied for more than one simulation iteration.

If the velocity ratio or bounciness b of an object is known the desired change in velocity Δv can be found:

$$\Delta v = v_2 - v_1 = -v_1 b - v_1 = -v_1(b + 1) \quad (12.6)$$

Then, the force needed to achieve this change in linear velocity for a given Δt can be found:

$$\Delta v = a\Delta t \quad (12.7)$$

$$a = \frac{f}{m} + g \quad (12.8)$$

$$\Delta v = \left(\frac{f}{m} + g \right) \Delta t \quad (12.9)$$

solving for f gives:

$$f = m \left(\frac{\Delta v}{\Delta t} - g \right) \quad (12.10)$$

and finally substituting:

$$f = -m \left(\frac{v_1(b+1)}{\Delta t} + g \right) \quad (12.11)$$

In summary, if this force f is exerted on an object moving at velocity v_1 with bounciness b for a duration of Δt in the direction away from the surface, the object should bounce realistically.

12.3 Bouncing in Three Dimensions: A Model for Polyhedral Object Bouncing

To simulate bouncing in three dimensions, The velocity of the vertex contacting the surface is separated into two components: one normal to the surface and one tangent to it. The velocity normal to surface is used to calculate the bounce force from the surface as in one dimension. The tangential component could be used to calculate friction, but is not in the current implementation.

When a polyhedral shape bounces on a surface, the force from the vertex contacting the surface is usually not aligned with the center of mass. The bounce will cause a torque to be exerted on the object as well as a linear force.

The kinetic energy of an arbitrary vertex of a polyhedron is more difficult to calculate than in the one dimensional case, because angular velocity and moment of inertia are involved as well as mass and linear velocity. How-

ever, the bounciness or desired velocity ratio can still be used to calculate a force that will cause the polyhedron to bounce on that vertex.

As shown above for linear bounces, a force normal to the surface can be calculated that will give a desired change in velocity for a given Δt . The contact velocity of a vertex that is at L relative to the center of mass of the polyhedron with translational velocity V_t and angular velocity V_θ is

$$V_1 = V_t + (V_\theta \times L) \quad (12.12)$$

If the normal of the surface is \hat{N} then the normal velocity component and the bounciness b give the desired scalar change in velocity in the normal direction is.

$$\Delta v = -V_1 \cdot \hat{N}(1 + b) \quad (12.13)$$

The strategy for finding the force that will cause the desired Δv will be to relate the acceleration and then Δv to a force F in the surface normal direction, and then solve the resulting equation for for the force magnitude f .

$$F = f\hat{N} \quad (12.14)$$

Acceleration of the vertex in the surface normal direction will be caused by both the translational and rotational effects of F on the polyhedron. The acceleration in the normal direction of the vertex due to polyhedron translational acceleration, with gravity G , is

$$a_{pt} = f/m + \hat{N} \cdot G \quad (12.15)$$

The moment of inertia i about the axis $L \times N$ is found as described previously, and the acceleration of the vertex due to polyhedron rotational acceleration is found:

$$a_{p\theta} = \hat{N} \cdot (L \times f\hat{N} \times L)/i \quad (12.16)$$

Then, the two accelerations above are summed and Δv in the surface normal direction is related to f :

$$\Delta v = a\Delta t \quad (12.17)$$

$$\Delta v = \left[N \cdot (L \times f\hat{N} \times L)/i + f/m + N \cdot G \right] \Delta t \quad (12.18)$$

Finally, solving for f gives:

$$f = \frac{\Delta v/\Delta t - \hat{N} \cdot G}{\hat{N} \cdot (L \times \hat{N} \times L)/i + 1/m} \quad (12.19)$$

So, if the force described by this final equation is exerted at the vertex at L over a time Δt , the polyhedron should bounce with reasonable rotational and translational dynamic accuracy.

Chapter 13

Sound

An intelligent animation system that provides efficiency in creating structure and motion, should also provide some efficiency in creating the related sound track. Complex structures can generate many sounds at very specific times. It should be easier for an animation system to generate these sounds than it is to create the sounds independently and then attempt to synchronize them to the animation. The animation system described in this thesis can detect and record the exact times of sound producing events.

13.1 Sound Events

A sound event description produced by the animation system contains the following information.

1. The frame number of the sound event.
2. Information about what caused the sound.
3. The location of the sound.

Sound events are created and collected when the frames of an animation are being generated. This information is then processed into MIDI time codes, where each sound event produces a note of arbitrary complexity. The frame number of the sound event determines the exact time of the note. The information about what caused the sound event, is used to determine the pitch and quality of the note. The location of the sound event could be used to affect the stereo balance and loudness of the note.

Each object in an animation may produce a specific type of sound, or the qualities of the objects could be used to determine the pitch and quality of the sound.

There are two categories of detectable sound events: collisions between objects, and motion of objects.

13.1.1 Collisions

An obvious example of a sound producing event is when two objects collide with each other. Falling objects can produce sounds when they bounce on a surface. As a creature walks or hops, each footstep can be detected, and sounds are produced as each foot makes contact with the terrain the creature is walking on. An example that would produce a lot of collision sound events would be a pile of wooden blocks crashing over. Inter object collisions as well as objects hitting the floor would each produce a wood like clunk, that together would give a crash.

13.1.2 Sounds from Motion

Sounds such as squeaks, squawks, whistles, swooshes, blorps, boings, and others that are difficult to describe, can all be caused by motions of objects that may not involve collisions. Some of these sounds, although rare in reality, are very useful in creating desired effects in animation. Bending joints can cause squeaks or squawks, objects speeding through the air could cause swooshes or whistles, and creatures pushing off the ground for a jump might cause boings.

Chapter 14

Conclusion

In summary, an animation system has been built that allows animation of arbitrary jointed-figures over uneven terrain with forbidden footstep locations. An interactive jointed figure editor is used to efficiently create creatures, that are analyzed to determine what parts are legs, body, head, or tail. Motions can be automatically generated of these creatures walking or pronking adaptively over the terrain.

Levels of motion representation have been built to create the complex motions of adaptive locomotion. Jointed figures are represented by a network of connected joints and links. Joints are initially described by their axis and position in a creation coordinate frame, and the links that they connect.

Forward kinematics or inverse kinematics can be performed on the limbs of a jointed figure. The pseudo-inverse of the jacobian matrix is used to solve the inverse kinematics problem iteratively on arbitrary limbs.

Dynamic simulation of single rigid bodies has been implemented and is

used to determine the body trajectory of pronking figures and the trajectories of bouncing polyhedra.

The terrain is represented in a way that allows fast surface height calculation and forbidden zone detection.

Locomotion parameters such as gait, bounciness, speed, and jump height, can be adjusted to vary the style of locomotion. Walking motions for jointed figures can be generated with arbitrary gait patterns using inverse-kinematics and step planning. Pronking can also be generated using dynamic simulation and inverse-kinematics.

Two other abilities of this animation system have been described, although they are not directly related to locomotion. Dynamic simulation and surface forces have been used to model bouncing polyhedra with linear and angular velocities. Sound event data can be recorded by the animation system for use in creating a well synchronized sound track.

Appendix A

Miscellaneous

A.1 Jointed Figure Editor Details

Here are some details not given in chapter 4 of the user interface that allows creation of jointed figures.

The interactive environment used for creating jointed figures consists of a menu of operations that can be chosen with a mouse, and a mouse sensitive area of the screen that allows drawing nodes and connections. Axes of the figure coordinate system are drawn in dotted lines so the figure can be aligned. The axes are sticky to allow perfectly straight legs: if a node is created near an axis it will be placed exactly on the axis.

Nodes that have been created are mouse sensitive, a temporary circle appears around them to indicate they are selected when the mouse is over them. A fair amount of functionality was needed from a mouse with only three buttons, so the function of each button depends on whether the mouse is over an existing node or not, and whether a connection between nodes has been initiated but not completed.

Below is a summary of the functionality of the mouse buttons for each possible state. The left button of the mouse will create a new node and a loose connection. A loose connection will cause a rubber band line to be drawn from the node to the cursor. This loose connection will form a complete connection with the next node created or clicked on. The middle button will create a new node without a connection, or it will delete an existing node if the cursor is on one. The right button will abort a loose connection.

	<i>No loose connection</i>		<i>Loose connection</i>	
	<i>Not over existing node</i>	<i>Over existing node</i>	<i>Not over existing node</i>	<i>Over existing node</i>
<i>Left</i>	create node and loose connection	no effect	create node and complete connection	complete connection
<i>Middle</i>	create node	delete node	create node and complete connection	complete connection
<i>Right</i>	no effect	no effect	abort connection	abort connection

The menu items allow the user to select a set of nodes that have been created and then perform operations on that selected set of nodes.

There are several methods for choosing a selected set of nodes. When a set of nodes is selected, those nodes and the connections between them are highlighted. The methods for choosing the selected set of nodes are:

1. *Select all.* This simply sets the selected set to include all nodes that have been created.
2. *Unselect all.* This sets the selected set to be empty.
3. *Reselect a previously selected set.* This loops through a ring of previously selected sets.

4. *Circle new set with mouse.* This allows the user to draw an arbitrary closed shape with the mouse. Any nodes that are inclosed by this shape become part of the new selected set. [See the next section of the appendix for determining the nodes inclosed by the shape.]

The selected set of nodes can then be operated on in several ways. It can moved or copied to a new position using a variety of transformations. *Delete* will remove the entire selected set and any connections attached to it. An *undelete* option will bring back the previously deleted set.

For any transformation the selected set can be moved to a new position, or copied to a new position, depending on which of *move* or *copy* is selected. One or the other is always selected [see figure 4.1]. The transformations available are *translate*, *rotate*, *rotate absolute*, *scale*, *scale x and y separately*, *mirror X*, and *mirror Y*.

A.2 Shape Enclosure Test

This is an easy way to determine if a point (p_x, p_y) is inside or outside an arbitrary complex 2-D shape. The shape may be concave, and it may be clockwise or counterclockwise. If the shape is defined by a set of n points (x_i, y_i) where the first point is repeated as the last one, and $atan2(y, x)$ gives the angle for any y and x , then

$$A_{sum} = \sum_{i=0}^{n-2} [atan2(y_{i+1} - p_y)(x_{i+1} - p_x) - atan2(y_i - p_y)(x_i - p_x)] \quad (A.1)$$

The angle sum should be either 2π , 0 , or -2π . If the angle sum is 2π or -2π the node is enclosed by the shape, if the angle sum is 0 it is not enclosed.

A.3 Rotating about an Arbitrary Angle

Given an axis vector $\hat{A} = (a_x, a_y, a_z)$, and a rotation angle θ , a matrix R can be created that will cause a rotation about \hat{A} by the amount of θ . First the axis is converted into spherical coordinates by finding $\text{atan}(a_y/a_z)$ and $\text{atan}(a_x/a_z)$. Then a rotation matrix Q is created with these spherical coordinates that will transform that axis to a point on the z axis. The inverse will transform the point from the z axis back to \hat{A} , and since Q is orthonormal, $Q^{-1} = Q^T$, which is much easier to compute. If a rotation about the z axis is described by the matrix $Z(\theta)$ then:

$$R = Q^T Z(\theta) Q \tag{A.2}$$

If rotation is repeatedly done about the same axis, Q and Q^T could be saved to increase efficiency.

Appendix B

The Pseudo-inverse

In this chapter, the pseudo-inverse for inverse kinematics is outlined. Methods for achieving secondary goals of singularity avoidance, avoiding joint limits, and obstacle avoidance are briefly described, and some methods for combining secondary goals are discussed. Most research of pseudo-inverse techniques has been done for the purpose of controlling robot manipulators. The same techniques apply directly to applications in computer graphics.

An inevitable problem in robot control is that of finding the joint behaviors from a cartesian task description of tip or end effector. For many manipulators direct inverse kinematics is possible and the joint angles θ can be calculated from a desired end effector position x . If a manipulator has more degrees of freedom than the dimension of x , it is called redundant, or underdetermined, and θ can not be calculated directly.

A common method for reaching a desired end effector position in redundant manipulators is using incremental velocity control. Although the joint velocities $\dot{\theta}$ that give a desired end effector velocity \dot{x} are not neces-

sarily unique, the solutions can be found using pseudo-inverse techniques. The velocities towards intermediate positions x_i of the final position are repeatedly calculated, and the corresponding joint velocities are updated [see figure 6.1].

B.1 The Jacobian

The Jacobian, J , is the matrix of partial derivatives that describes the change in cartesian coordinates due to the change in each joint angle [See chapter 5]. J is a function of the arm's structure and current joint angles.

For a vector of joint velocities $\dot{\theta}$, the vector of cartesian velocities \dot{x} is given by

$$\dot{x} = J\dot{\theta} \quad (\text{B.1})$$

The desired form of this relationship is

$$\dot{\theta} = J^{-1}\dot{x} \quad (\text{B.2})$$

that is, given a desired cartesian velocity, find a joint velocity. But J^{-1} is only defined when J is square and non-singular. When a manipulator is either overdetermined or underdetermined J is rectangular. If \dot{x} is m dimensional, and $\dot{\theta}$ is n dimensional, J is an $m \times n$ matrix. In the case of redundancy there are a range of $\dot{\theta}$ that satisfy $\dot{x} = J\dot{\theta}$. The smallest $\dot{\theta}$ that satisfies this equation is given by

$$\dot{\theta} = J^+ \dot{x} \quad (\text{B.3})$$

where smallest refers to the minimum norm solution, and J^+ is the pseudo-inverse of J . For redundant situations, the pseudo-inverse of J can be calculated by

$$J^+ = J^T (J J^T)^{-1} \quad (\text{B.4})$$

Clever methods of computing $J^+ \dot{x}$ without actually computing J^+ itself, such as using LU decomposition, have been pointed out [15], the method used in this work was developed by [58] and implemented by Alejandro Ferdman [9].

The null space of J , $N(J)$, contains all the vectors $\dot{\theta}_n$ that map into $\dot{x} = 0$, that is, $J \dot{\theta}_n = 0$. They are called the homogeneous solutions. For a given \dot{x} we can add any $\dot{\theta}_n$ in $N(J)$ to the minimum norm solution, and the resulting $\dot{\theta}$ will still give the requested \dot{x} .

$$\dot{\theta} = J^+ \dot{x} + \dot{\theta}_n \quad (\text{B.5})$$

Any vector z in $\dot{\theta}$ space, can be projected onto $N(J)$ by the projection matrix $(I - J^+ J)$. So,

$$\dot{\theta} = J^+ \dot{x} + (I - J^+ J) z \quad (\text{B.6})$$

for all z , describes the sub-space of $\dot{\theta}$ space that satisfies $\dot{x} = J \dot{\theta}$. This sub-space is parallel to $N(J)$ since it is just $N(J)$ offset by the minimum

norm solution. For a given z , the resulting $\dot{\theta}$ will be the minimum norm solution combined with the closest vector to z that does not change the corresponding \dot{x} [see figure B.1].

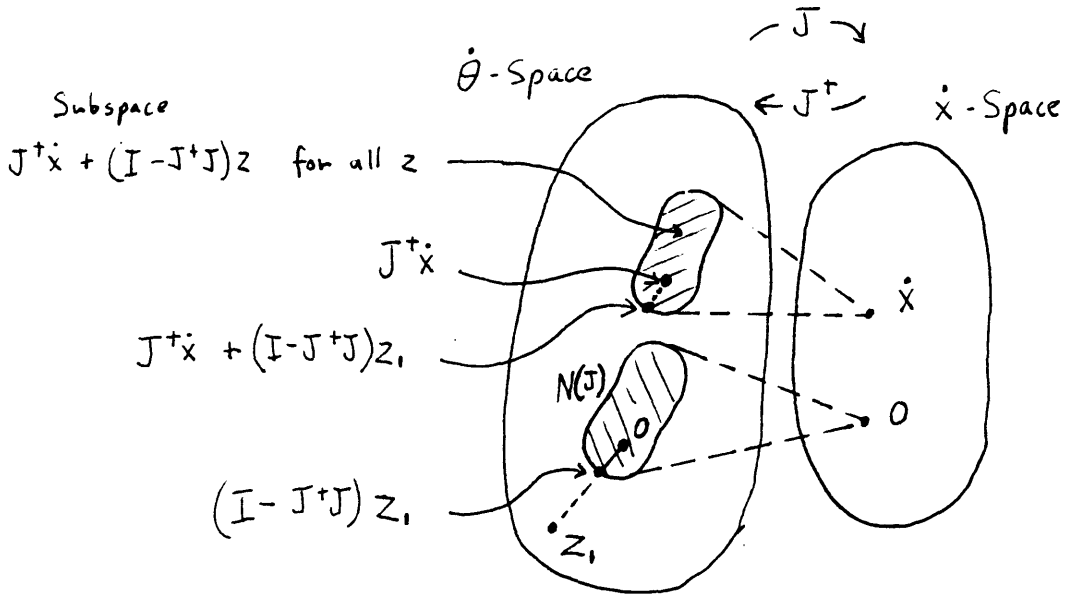
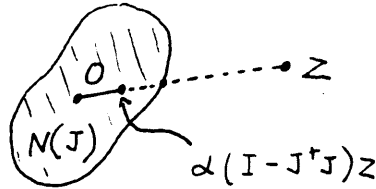


figure B.1

B.2 Secondary Goals

Some secondary goals, such as avoiding singularities, and avoiding joint limits allow a single desired joint velocity z to be calculated and used. For these cases the formula above can be used. A gain coefficient α can be added to give a variable weight to the secondary goal [see figure B.2].

$$\dot{\theta} = J^+ \dot{x} + \alpha(I - J^+J)z \tag{B.7}$$

figure B.2 $\alpha = 0.5$ 

If some function measures the desirability of an arm configuration, the gradient of this function gives the best value of z . For singularity avoidance this function measures the dexterity, or how able the arm is to move in all directions. Near a singularity the dexterity approaches zero. Much work has involved evaluating different methods for measuring manipulator dexterity and its gradient as a function of θ [24,41,74]. If there is a set of preferred joint angles for the manipulator, z could be simply the difference between the current and preferred joint angles.

If z_d is the desired total joint velocities, as opposed to velocities to be added to the minimum norm solution, then

$$z = z_d - J^+ \dot{x}, \quad (\text{B.8})$$

$$\dot{\theta} = J^+ \dot{x} + \alpha(I - J^+ J)(z_d - J^+ \dot{x}) \quad (\text{B.9})$$

Other secondary goals such as obstacle avoidance, can have a range of acceptable joint velocities. When this range is projected onto the sub-space

of the primary goal solutions, an intersection space is possible, signifying some remaining redundancy [see figure B.3].

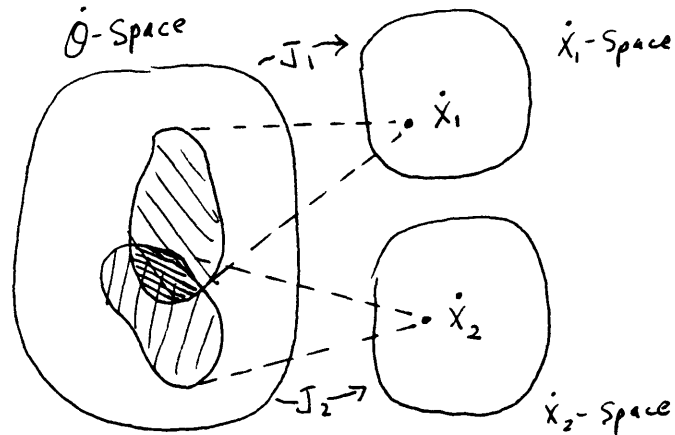


figure B.3

If a certain link is too close to an obstacle, it should move away from it if possible. The joints between that link and the end effector have no effect on moving that link. Another Jacobian J_0 describing the arm up to that link, and the cartesian velocity \dot{x}_0 that moves that link away from the obstacle can be used to give another sub-space of $\dot{\theta}$ space. The intersection of these sub-spaces corresponds to all $\dot{\theta}$ that satisfy both $\dot{x} = J\dot{\theta}$ and $\dot{x}_0 = J_0\dot{\theta}$. Some of these projections, and intersections are formulated in [24,28,29,41].

B.3 Combining Goals

A set of goals can be combined using prioritization, weighting, or by a mixture of the two. Goals with ranges of equally acceptable solutions lend themselves to prioritizing, while goals that give a single preferred solution are better suited to being weighted.

Prioritizing causes a goal of lower priority not to lower the measure of success of any goal of higher priority. Each sub-space is projected onto $N(J)$ in turn until the redundancy of the arm is used up [29]. This works well if the manipulator is highly redundant, but if there are few degrees of redundancy, and many prioritized goals, the lowest priority goal may never be considered. Once the redundancy is used up no lower priority goals have an affect.

Weighting is equivalent to combining several goals into a single goal that changes focus as the arm configuration changes. When secondary goals are weighted, they can affect each other. A method of performing weighted control would be to combine several desired joint velocities z_i each with a weight w_i , to give a final vector z . Both z_i and w_i would probably be functions of θ .

$$z = w_1 z_1 + w_2 z_2 + w_3 z_3 \dots \quad (\text{B.10})$$

Weighting should be useful when several goals are similarly important, but each only necessary at certain arm configurations. It allows each goal to have an effect sometimes, even if the degree of redundancy is low. Groups

of weighted goals could be prioritized absolutely as above.

Appendix C

Acknowledgments

Many thanks to:

David Zeltzer, my advisor, for seeing into the future of computer animation, and for great Rball games. Steve Strassman, for keeping the lisp machines working, and for general hacker companionship. Alejandro Ferdman for providing a foundation for this work with his pseudo-inverse code. Jim Salem, for help with the 3d Image Toolkit which was used for the rendering in this thesis. Marc Raibert, for ideas and encouragement. Marvin Minsky, for believing in AI. Alexandra, for love and friendship. Allison, for energy, friendship, and support. Clea, for memories. Pascal, for adjusting video levels. Mike and Necco, for being fine office mates. Paul, for always having what you need. Dave Ross, for putting up with us complaining about his box. And to all the folks in the garden, for entertainment, distractions, and for being a great bunch.

Bibliography

- [1] Alexander, R. McN., and Vernon, A. "The Mechanics of Hopping by Kangaroos (Macropodidas)", *Journal of Zoology*, 1975, London, vol 177, pp 265-303.
- [2] Armstong, W.W., and Green, M. "The Dynamics of Articulated Rigid Bodies for Purposes of Animation" *Graphics Interface*, 1985.
- [3] Asmussen, E. "Movement of man and study of man in motion: a scanning review of the development of biomechanics", International Series on Biomechanics, Vol 1A, Biomechanics V-A, Proceedings of the Fifth International Congress of Biomechanics, Jyvaskyla, Finland, pp 23.
- [4] Badler, Norman, and Smoliar, Stphen, "The Representation Human Movements Using a Digital Computer", October 1977, Moore School of Science, University of Pennsylvania, Philadelphia, MS-CIS-78-4.
- [5] Badler, Norman, "A Representation for Natural Human Movement", Tech. Report MS-CIS-86-23, Dept. of Computer and Information Science, University of Pennsylvania Philadelphia, PA, 1986.
- [6] Badler, Norman, "Design of a Human Movement Representation Incorporating Dynamics." Course Notes, Seminar on Three-Dimensional Computer Animation, *ACM siggraph*, 1982.
- [7] Badler, Norman I., Dorein, Johnathan D., Korein, James U., Radack, Gerald M., and Brotman, Lynne Shapiro, "Positionaing and Animating Human Figures in a Task Oriented Environment" *The Visual Computer*, 1985, pp 212-220.

- [8] Cavagna, G.A., Heglund, N.C., and Taylor, C.R., "Mechanical work in Terrestrial Locomotion: Two basic Mechanisms for minimizing energy expenditure." *American Journal of Physiology*, 1977, vol 233, pp 243-261.
- [9] Ferdman, Alejandro J, "Robotics Techniques for Controlling Computer Animated Figures", M.S.Vis.S. Thesis, M.I.T., August 1986.
- [10] Frank, A.A., "On the Stability of an Algorithmic Biped Locomotion Machine", *Journal of Terramechanics*, 1971 vol. 8 no 1. pp 41-50
- [11] Gallistel, C.R., *The Organization of Action*, Lawrence Erlbaum Associates, Publishers, 1980 Hillsdale, NJ
- [12] Ginsberg, C., and Maxwell, D., "Graphical Marionette," *Proceedings, AMC siggraph/sigart workshop on motion*, April 1983, pp. 172-179.
- [13] Girard, Michael, and Maciejewski, A.A., "Computational Modeling for the Computer Animation of Legged Figures", *Siggraph proceedings* 1985.
- [14] Girard, Michael, "Interactive Design of 3-D Computer-Animated Legged Animal Motion", ACM, October 1986.
- [15] Greville, T.N.E., "Some applications of the pseudoinverse of a matrix", *SIAM Review*, vol 2, no 1, 1960
- [16] Gubina, F., Hemani, H., and McGhee, R.B., "On the dynamic stability of biped locomotion" *IEEE Transactions on Biomedical Engineering*, 1974, BME-21, pp 102-108.
- [17] Herbison-Evans, Don, "A Human Movement Language for Computer Animation" *Proceedings of the symposium on Language Design and Programming Methodology*, Sydney, 10-11, September 1979, pp 117-127 from *Lecture Notes in Computer Science* edited by G. Goos and J. Hartmanis.
- [18] Hildebrand, Milton, "How Animals Run", *Scientific American*, 1960, pp 148-157.

- [19] Hildebrand, Milton, "Analysis of Tetrapod Gaits: General Considerations and Symmetrical Gaits", *Neural Control of Locomotion*, edited by R.N.Herman, S. Grillner, PsS. Stein, and D.G. Stuart, Plenum Publishing Corporation New York, NY, 1976, pp 203-236.
- [20] Hill, A.V. "The dimensions of animals and their muscular dynamics" *Science Progress*, 1950, vol XXXVIII, pp 209-230.
- [21] Hirose, S., and Umetani, Y. "The basic motion regulation system for a quadruped walking vehicle", *ASME Conference on Mechanisms*, 1980.
- [22] Hoyt, Donald F. and Taylor, Richard C., "Gait and the Energetics of Locomotion in Horses", *Nature*, Vol 292, July 16, 1981, pp 239.
- [23] Inman, Verne T., Ralston, Henry J., Todd, Frank, *Human Walking*, Williams and Wilkins Baltimore/London, 1980.
- [24] Klein, A. Charles, "Use of Redundancy in the Design of Robotic Systems", p208 - 214
- [25] Klein, C.A., and Briggs, R. L., "Use of Active Compliance in the control of legged vehicles" *IEEE transactions of Systems, Man, and Cybernetics*, 1980, vol SMC-10, pp 393-400.
- [26] Klein, A. Charles, and Huang, Ching-Hsiang, "Review of Pseudoinverse Control for Use with Kinematically Redundant Manipulators", *IEEE transactions*, vol SMC-13 no 3, March/April 1983, p254 - 250
- [27] Lee, C.S.George, "Robot Arm Kinematics, Dynamics, and Control", IEEE 1982
- [28] Maciejewski, Anthony Alexander, "Obstacle Avoidance for Kinematically Redundant Manipulators", Thesis, Ohio State University, March 1984.
- [29] Maciejewski, Anthony A. and Klein, A. Charles, "Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments", *The International Journal of Robotics Research*, vol 4, no 3, 1985, p109 - 117

- [30] McGhee, Robert B. "Some Finite State Aspects of legged Locomotion" *Mathematical Biosciences*, 1968, vol 2, pp 67-84.
- [31] McGhee, Robert B., and Frank, A.A., "On the stability properties of quadruped creeping gaits" *Mathematical Biosciences*, 1968, vol 3, pp 331-351.
- [32] McGhee, Robert B. "Robot Locomotion", *Neural Control of Locomotion*, Herman, et. al, New York, Plenum Press, 1976.
- [33] McGhee, Robert B. and Iswandhi, Geoffrey I., "Adaptive Locomotion of a Multilegged Robot over Rough Terrain", *IEEE transactions on systems, man, and cybernetics*, vol 9, no. 4, april 1979, p176 - 182
- [34] McMahan, T.A., "Using body size to understand the structural design of animals: Quadrupedal locomotion." *Journal of Applied Physiology*, vol 39, pp 619-627.
- [35] McMahan, T.A., "The Role of Compliance in Mammalian Running Gaits", *Journal of Experimental Biology*, 1985, vol 115, pp 263-282.
- [36] Melvill-Jones, G. and Watt, D.O.D., "Observations on the control of stepping and hopping movements in man", *Journal of Physiology*, 1971, vol 219, pp 709-727.
- [37] Messuri, Dominic A. and Klein, Charles A., "Automatic Body Regulation for Maintaining Stability of a Legged Vehicle During Rough-Terrain Locomotion", *IEEE Journal of Robotics and Automation*, Vol RA-1, No 3, September 1985, pp 132 - 140.
- [38] Minsky, Marvin, *The Society of Mind*, Simon and Schuster, New York, 1986.
- [39] Muybridge, E. 1955. *The Human Figure in Motion*, New York: Dover publications. First edition, 1901 by Chapman and Hall, Ltd. London.
- [40] Muybridge, E. 1957. *Animals in Motion*, New York: Dover publications. First edition, 1899 by Chapman and Hall, Ltd. London.
- [41] Nakamura, Yoshihiko, and Hanafusa, Hideo, "Task Priority Based Redundancy Control of Robot Manipulators", *2nd International Symposium on Robotics Research*, MIT Press, 1985, pp 155 - 162.

- [42] Nashner, L.M. "Balance adjustments of humans perturbed while walking", *Journal of Neurophysiology*, vol 44, pp 650-664.
- [43] Orin, David E., and Schrader, William W. "Efficient Computation of the Jacobian for Robot Manipulators" *The International Journal of Robotics Research*, Vol. 3, No. 4, Winter 1984, pp 66-75.
- [44] Paul, Richard P., *Robot Manipulators*, MIT Press, Cambridge, Mass., 1981, p 54.
- [45] Paul, Richard P., Shimano, Bruce, and Mayer, Gordon E. "Differential Kinematic Control Equations for Simple Manipulators", *IEEE transactions on systems, man, and cybernetics*, vol SMC-11, no. 6, June 1981, pp 456-460.
- [46] Pearson, K.G. "The control of walking", *Scientific American*, 1976, 72-86.
- [47] Pearson, K.G., and Franklin, R., "Characteristics of leg movements and patterns of coordination in locusts walking on rough terrain", *International Journal of Robotics Research*, vol 3, pp 101-112.
- [48] Raibert, Marc H., Brown, H.B.Jr., "Experiments in Balance with a 2D One-Legged Hopping Machine", *ASME Journal of Dyn. Sys. Measurement, Cont.* vol 106, pp 75-81.
- [49] Raibert, Marc H., Brown, H. Benjamin Jr., Chepponis, Michael, "Experiments in Balance with a 3D One-Legged Hopping Machine", *International Journal of Robotics Research*, vol 3, no 2, summer 1984, pp 75-92.
- [50] Raibert, Marc H., "Symmetry in Running", *Science*, 1986, vol 231, pp 1292-1294.
- [51] Raibert, Marc H., *legged robots that balance*, Cambridge: MIT Press, 1986.
- [52] Raibert, Marc H., Brown, H.B. Jr., "Running on four legs as though they were one", *IEEE Journal of Robotics and Automation*, vol 2, pp 70-82.

- [53] Raibert, Marc H., "Running with Symmetry", *International Journal of Robotics Research*, vol 5, no 4, winter 1987, pp 3-19.
- [54] Reeves, W., "Partical Systems - a technique for modelling a class of fuzzy objects" *Computer Graphics* 18,3, July 1983, pp 359-376.
- [55] Reeves, W., and Blau, R. "Approximate and probabilistic algorithms for shading and rendering structured particle systems", *Computer Graphics* 19,3, July 1985, pp 313-322.
- [56] Reynolds, Craig W., "Computer Animation with Scripts and Actors" *Computer Graphics*, Vol 16, No 3, July 1982.
- [57] Reynolds, Craig W., "Flocks, Herds, and Schools, A Distributed Behavioral Model", to be published, siggraph proceedings 1987.
- [58] Ribble, E.A., "Synthesis of Human Skeletal Motion and the Design of a Special-Purpose Processor for Real-Time Animation of Human and Animal Figure Motion", M.S. Thesis, The Ohio State University, June 1982.
- [59] Rogers, David F., *Procedural Elements for Computer Graphics*, McGraw Hill, New York, 1985.
- [60] Saltzman, Elliot, "Levels of Sensorimotor Representation", *Journal of Mathematical Psychology*, vol 20, pp 91-163, 1979.
- [61] Severin, F.V., Orlovskii, G.N. and Shik, M.L., "Work of the Muscle Receptors During Controlled Locomotion"
- [62] Taylor, C.R., and Rowntree, V.J., "Running on two of four legs: Which consumes more energy?" *Science*, vol 179, pp 179-187.
- [63] Thalmann, Nadia Magnenat, and Thalmann, Daniel, "The Use of High-Level 3-D Graphical Types in the Mira Animation System", IEEE 1983, pp 9-14.
- [64] Thalmann, Nadia Magnenat, and Thalmann, Daniel, "Three-Dimensional Computer Animation: More an Evolution Than a Motion Problem", IEEE 1985, pp 47-57.

- [65] Thomas, Frank, "Can classic Disney animation be duplicated on the computer?", *Computer Pictures*, July/Aug. 1984, pp 20-26.
- [66] Waldron, Kenneth J. and Kinzel, Gary L., "The Relationship between Actuator Geometry and Mechanical Efficiency in Robots", *Theory and Practice of Robots and Manipulators, Proceedings of RoManSy*, 1981, edited by A. Morecki, G. Bianchi, K. Kedzior, Warsaw: Polish Scientific Publishers, pp 305-316.
- [67] Waldron, Kenneth J. and Vohnout, Vincent J. "Configuration Design of the Adaptive Suspension Vehicle", *The International Journal of Robotics Research*, Vol. 3, No. 2, Summer 1984.
- [68] Walker, M.W., and Orin, D.E. "Efficient Dynamic Computer Simulation of Robotic Mechanisms" *Journal of Dynamic Systems, Measurement, and Control*, September 1982, vol. 104, p205.
- [69] Warren, W.H.Jr., Young, D.S., and Lee, D.N., "Visual control of step length during running over irregular terrain", *Journal of experimental Psychology*, vol 12, pp 259-266.
- [70] Whitney, D. E., "The Mathematics of Coordinated Control of Prosthetic Arms and Manipulators", *Journal of Dynamic Systems Measurement, and Control*, December 1972 pp 303-309.
- [71] Williams, L, "BBOP" Course Notes, Seminar on Three Dimensional Computer Animation, *ACM siggraph*, 1982.
- [72] Wilhelms, Jane, and Barsky, A.B., "Using Dynamic Analysis to Animate Articulated Bodies Such As Humans and Robots" *Graphics Interface 85 proceedings*, Montreal, Canada, May 1985, pp 97-115.
- [73] Wilson, Donald M., "Insect Walking", *Annual Review of Entomology*, 1966, vol 11, pp 103-121.
- [74] Yoshikawa, Tsuneo, "Analysis and Control of Robot Manipulators with Redundancy", *1st International Symposium on Robotics Research*, MIT Press, 1984, pp 735-747.
- [75] Zeltzer, David, and Csur, C., "Goal-Directed Movement Simulation", *Proceedings - Canadian Society for Man-Machine Interaction*, June 1981, pp 271-279.

- [76] Zeltzer, David, "Motor Control Techniques for Figure Animation", *IEEE Computer Graphics and Applications*, November 1982, pp 53-49.
- [77] Zeltzer, David, "Towards an integrated view of 3-D computer animation", *The Visual Computer*, December 1985, vol 1, pp 249-259.
- [78] Zeltzer, David, "Motor Problem Solving for Three Dimensional Computer Animation", to be published, *Proceedings L'Imaginaire numerique*, France, May 1987.