Footprint-based Quadruped Motion Synthesis

by

Nick Torkos

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Footprint-based Quadruped Motion Synthesis

Nick Torkos

Master of Science

Graduate Department of Computer Science

University of Toronto

1997

Abstract

Footprints are data structures that specify the positions, orientations, and timing information for the feet of articulated figures. This thesis studies the use of footprints to generate animations of legged animals. After introducing several methods to generate footprints, we develop an algorithm that synthesizes realistic motions of animals from footprints only. Our method employs a hybrid physically-based/kinematic approach. We first perform a trajectory optimization on an underlying approximation of the animal, in order to determine its overall motion. Soft physical and kinematic constraints employed in the optimization functional ensure that the trajectory's motion is both physically plausible and visually pleasing. By utilizing the optimized trajectory information, the footprint positions, and a variety of animation techniques, we reconstruct the motion of the animal. Our results show that various motions can be synthesized with this approach, including walking, galloping, jumping, pushups, and skating motions. Finally, we summarize our contributions and discuss future directions.

Acknowledgements

I would like to thank my supervisor, Michiel van de Panne, for his excellent supervision and guidance. He developed the underlying concepts on which this work was based, and inspired my creativity with his great brain storming sessions. Thanks, Michiel!

Eugene Fiume, who enthusiastically read over my thesis, helped me look at my work from a different perspective, and gave me some ideas to speed up my animation routines.

My friend and fellow Master's student, Ivan Neulander, for allowing me to use his paintstroke rendering system to display my animated creature in all its hairy glory. The paintstroke primitive in that system made it easy for me to generate images, and the A-buffer implementation in the renderer kept the images crisp and anti-aliased.

My friends at DGP and DCS. You have all made my experience as a graduate student unforgettable. Thanks for distracting me from the pressures of research.

Finally, I would like to thank my parents, George and Freda, and my brother, Jim. Without their encouragement and support, I would not have gotten as far as I have in my education and personal development. I want them to know that I appreciate all they have ever done for me.

Contents

List of Figures

Chapter 1

Introduction

Animals have always been a popular subject for animation. So popular in fact, that many of the great moments in animation history often involve animating animals. One of the first cartoon characters ever created was a dinosaur called Gertie who interacted with its creator, Winsor McCay [37]. In 1925, Willis J. O'Brien used stop-motion animation to reproduce a savage prehistoric earth ruled by dinosaurs in the *Lost World* [45]. The trend has continued in the realm of computer graphics. *Jurassic Park* [16] was a groundbreaking movie that used computer generated characters (dinosaurs) as key players in the film. Computer animation is currently a common and popular method for animating animals on film. Recent representative examples include the likes of *Jumanji* [14] and *Dragonheart* [47], and television shows such as *Hercules: The Legendary Journies* [46]. There is also a need for the animation of legged characters in interactive games. The Imperial walkers in *Star Wars: Shadows of the Empire* [41] are one such example.

It should be no surprise that computer animation is often the method of choice for putting animals on film. In some cases, the animal is mythical or extinct, and in others it is often difficult or even impossible to control the animal to do what the director wants. Computer animation simplifies the animation process by allowing the animator to manipulate and fine-tune the animation before the final print is made. However, the job of the animator has not really changed since animators still use keyframes for most of the animation that they do, only now they use a slightly different set of tools. Thus, animations remain difficult to generate and require highly skilled animators to make

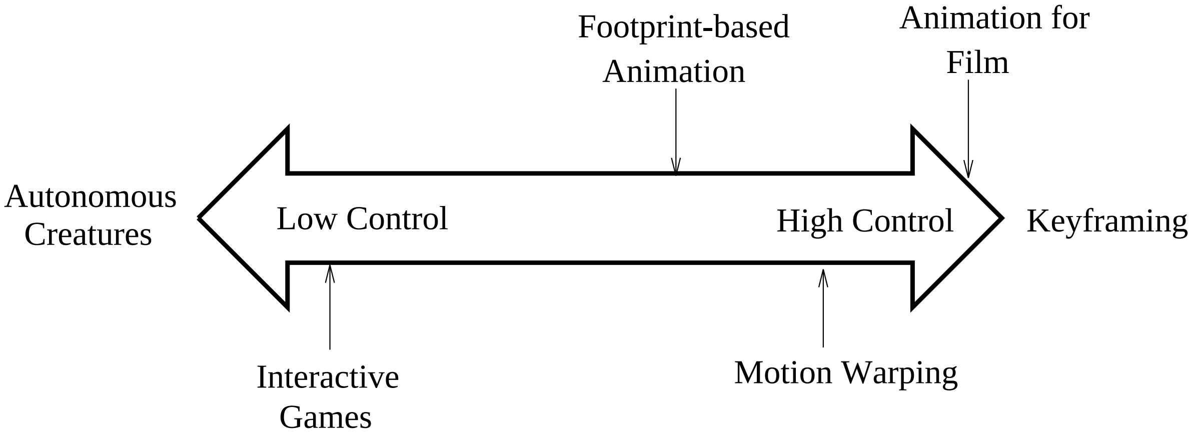


Figure 1.1: Graph showing where footprint-based animation fits within the range of control techniques.

them. As the demand for computer animation increases, it becomes more important to develop a set of tools that can remove the tedious aspects of animation. Currently, the most popular method of creating animations, *keyframing*, involves specifying a set of key poses for the articulated figure and then letting the computer generate the intermediate frames. Keyframing can be tedious since it requires the animator to position the joints for each keyframe by hand. In our opinion, an ideal tool would let animators quickly define a creature, specify its pose and path in an intuitive fashion, and then let the computer generate the details of the animation. We have a method that does this using footprints as animation primitives for locomotion. As shown in Figure 1.1, our method still allows the animator to control key characteristics of the locomotion, while using various algorithms to synthesize many of the details.

1.1 Motivation

Both people and legged animals leave trails of footprints on the ground when walking along an impressionable surface. As such, we all have some understanding of how footprints are related to the motions that created them. A computational method that can generate realistic animations from footprints is therefore a potentially powerful animation tool for use by animators of all levels of experience.

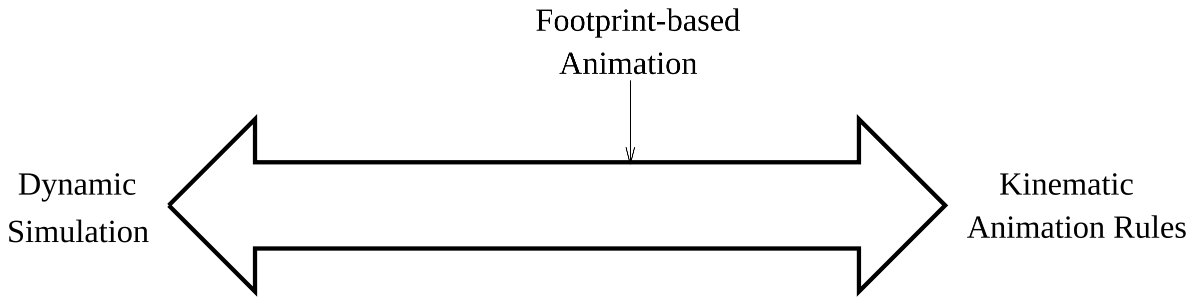


Figure 1.2: Graph showing where footprint-based animation ranks on the dynamics/kinematics scale.

How can the motion of an animal be reconstructed from a set of footprints? The main difficulty in answering this question arises from the fact that footprints only constrain the feet of the animal. To realize a realistic animation, all of the unconstrained body parts must be animated in a way that is both physically realistic and visually pleasing.

The use of physics is essential in the process of motion synthesis because footprint positions and times might necessitate ballistic motions. However, the amount of physical simulation required is debatable. Full physical simulations of articulated animals is at best of $O(n)$ complexity in the number of links for each time step, but in practice it remains a computationally expensive proposition.

Although full simulations take care of the motion of every joint, they introduce the harder problem of control. Any movement in one limb has repercussions in the motion of the other limbs, so great care must be taken in coordinating the amount of force that the muscles exert. This has limited the usage of true physical simulations to only simple models.

Our belief is that it is not necessary to implement physics throughout the motion synthesis in order to generate a natural looking animation. We wish to apply a hybrid approach to motion synthesis: one that utilizes a variety of techniques, including the use of a simplified physics model, and applies them where they can be the most advantageous. Thus we avoid applying computationally expensive techniques, such as physically-based modeling, to areas where they do not strongly influence the motion. Figure 1.2 illustrates where our footprint-based approach fits within the range of animation techniques that

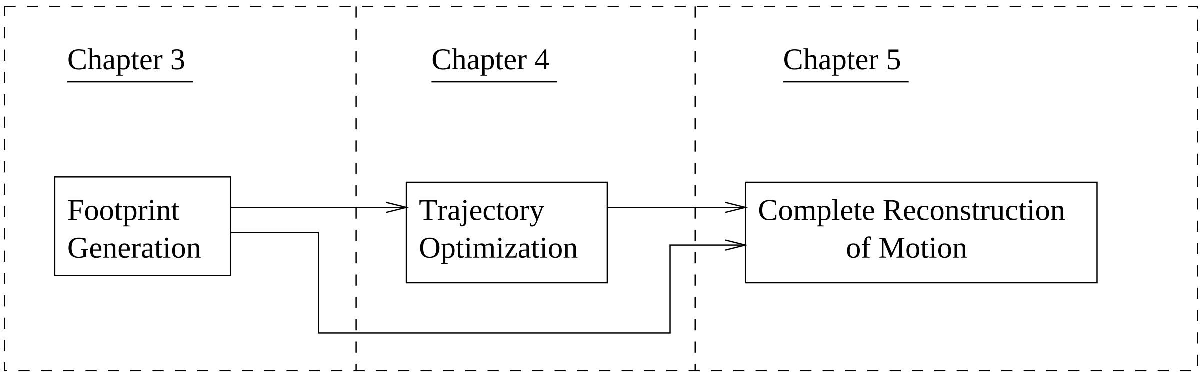


Figure 1.3: Overview of steps required to generate an animation with our system.

utilize dynamic simulation or kinematic animation rules.

1.2 Overview of Method

Figure 1.3 shows a graphical model of the steps required to generate an animation with our method, as well as where they are described in the thesis. The arrows show the dependencies among the steps

For the purposes of animation, we define a *footprint* as a set of parameters containing the foot's position, orientation, time of initial contact with the ground, and time of departure. Footprint generation encompasses all methods for generating footprints which serve as input for our animation method. The next two steps synthesize the motion of the animal. Figures 1.4 and 1.5 show a quadruped model, which will be used throughout the thesis to illustrate our algorithms. A hybrid physically-based/kinematic model generates the animation of this skeleton over the footprints.

During the trajectory optimization step of the motion synthesis, the model in Figure 1.6 is used as a coarse approximation of the true physical model. The body is approximated by two point masses and an interconnecting spring. The motions of the two point masses is represented by spline-based trajectories, and are used as a basis for the underlying motion of the quadruped. Each trajectory is subject to an optimization that ensures the resulting motion of this simplified model is both physically plausible and visually pleasing.

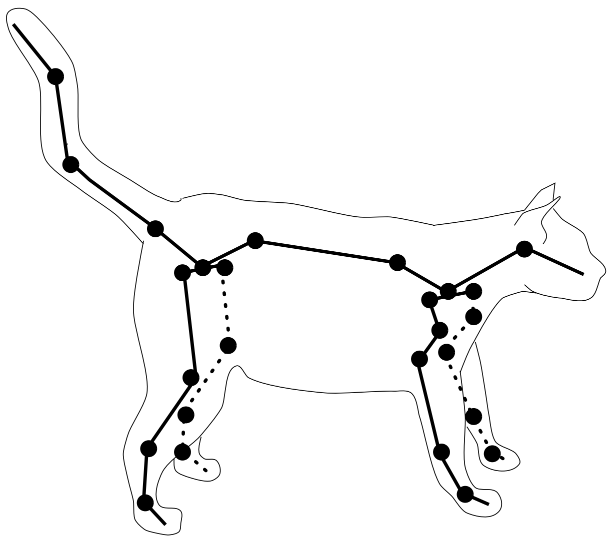


Figure 1.4: Initial model: side view

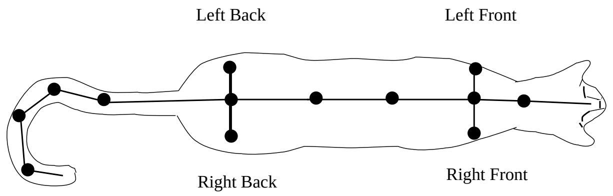


Figure 1.5: Initial model: top view

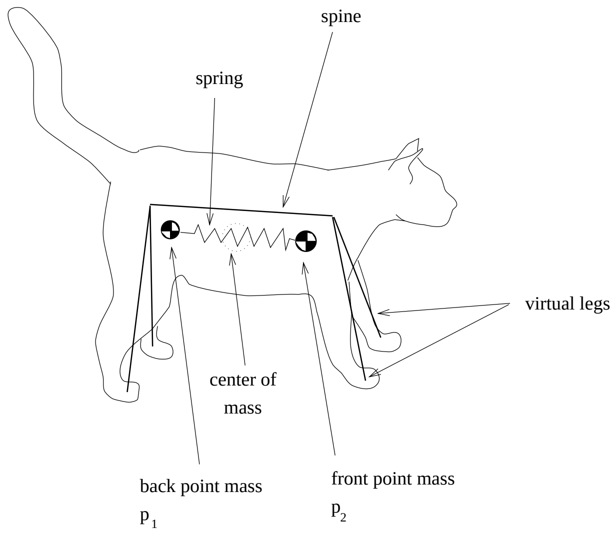


Figure 1.6: Simplified physical model.

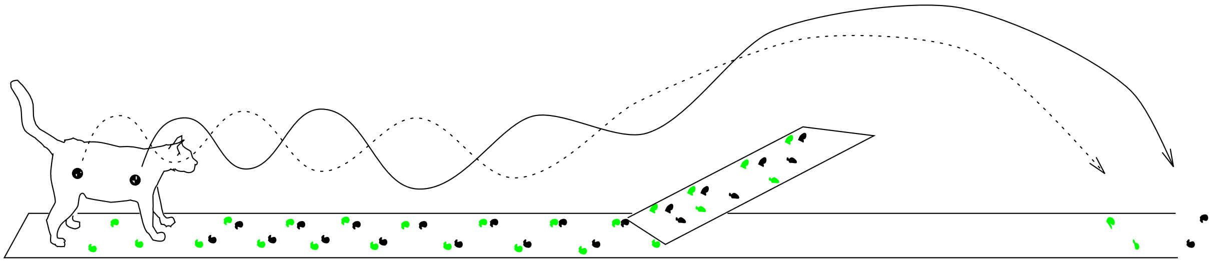


Figure 1.7: Trajectory optimization over a set of footprints.

Our simplified model allows us to check the physical feasibility of the point mass accelerations during the movement as defined by the spline trajectories. Note that this is not the same as examining static support, as commonly proposed in dedicated walking algorithms [12], because we do not require static stability.[1] The visual appeal of the trajectory is quantified by a *comfort* term, which is effected by a penalty that calculates the quadruped deviation at any time with respect to a range of allowable positions. A typical result of the trajectory optimization is a pair of trajectories as shown in Figure 1.7.

Reconstructing the full motion from the trajectories requires constraining known degrees of freedom until the skeleton is fully constrained. We use the known footprint locations to compute desired bends at the shoulders and hips. The leg positions are further constrained using a flexible parameterized inverse-kinematic model. Finally, secondary passive motion is employed to compute the motions of the tail and fur. We wish to emphasize that there may be many other suitable reconstruction methods. The most important concept about the reconstruction phase is that the freedom exists to choose whatever combination of methods produces the best results and is most convenient.

1.3 Thesis Contributions

This thesis develops a novel method for animating quadrupeds on footprint information alone. The proposed method is unique in its integrated use of dynamic and kinematic models. It explicitly allows for animated motions which strike a compromise between physical and animator constraints.

[1] Static stability requires that the center of mass should project within the support polygon made by the feet.

We apply the various techniques available where they are best suited, and avoid using methods where they would present us much difficulty. Our work also introduces a generic simplified dynamics model, and the application of *comfort* penalties - numerical calculations evaluating how awkward the creature looks - to guide the motion. Finally, we introduce a parameterized example-based inverse kinematics model for the legs during the reconstruction phase of our method.

1.4 Thesis Overview

The remainder of this thesis is divided into 6 chapters and 3 appendices. In Chapter 2 we discuss previous work on articulated figure animation. Our interface for generating the footprint data is presented in Chapter 3. The process of trajectory optimization is explained in Chapter 4. In Chapter 5, we describe how the motion of the animal is reconstructed from the trajectory information. Chapter 6 presents our results. Finally, we present our conclusions in Chapter 7 and include suggestions for future work. The appendices describe in detail several components of our particular implementation.

Chapter 2

Background

Motion synthesis is an interdisciplinary field which includes work from areas such as animation, robotics and biomechanics. This chapter summarizes previous work in the computer animation of articulated figures as well as some related areas where the motion of articulated figures is studied.

2.1 Basic Definitions

There are two fundamentally different methods to generate the motion for computer animation: *kinematics* or *dynamics*. An animation has been generated with *kinematics* if the motion is created without taking into account the forces involved in moving the bodies. Calculating the end position of an articulated limb when given the joint angles is called *forward kinematics*. The reverse process of determining a set of joint angles from an end effector position is called *inverse kinematics*. Most animations produced today use kinematic methods because it is typically easier to control kinematic animations than it is to control dynamic ones.

If all of the motion occurs as a direct result of forces applied to the body being animated then we say that *dynamics* is used in the animation. Newtonian or Lagrangian mechanics can be used to calculate the motion of rigid bodies from the forces applied on them. If the applied forces are known and used to solve for the unknown accelerations then we call this *forward dynamics*. Calculating the forces required to generate a specific motion is called *inverse dynamics*.

Kinematics and dynamics may be combined into *hybrid* animation methods. Hybrid schemes have the potential to utilize the relative advantages of both kinematics and dynamics.

2.2 Kinematic Methods

2.2.1 Keyframe Animation

Keyframing is the natural extension of traditional animation to the computer. As in traditional animation involving drawings or clay figures, computer keyframing requires the user to explicitly specify the joint positions of an animated figure at discrete instances in time. However, the computer offers the animator more flexibility in manipulating the figures than the traditional methods do. One example is the computer's ability to smoothly interpolate between adjacent keyframes. An outline of some issues involved in interactive keyframing is discussed by Sturman [56]. Because keyframing gives the user complete control over the motion of the articulated figure from frame to frame, it remains a commonly used technique in computer animation. Many of the creatures in the movie Jurassic Park [16], for example, were animated using stop-motion animation with the help of a device by Knep et al. called the Dinosaur Input Device [29], which captured the joint positions of a skeletal model. Although methods to simplify keyframing techniques have been developed [64], keyframing remains tedious.

2.2.2 Procedural Animation

Rather than having the user specify keyframes of the articulated figure, specific algorithms can be defined to generate the keyframes for a motion. One example is the JACK system by Badler et al. [4]. The purpose of JACK is to create an interactive computer graphics human model that can be used for various simulations. This is accomplished by capturing various human characteristics through simple behavioral laws discussed by Badler and Phillips in [5]. Among other things, the articulated figures in the JACK system can reposition their bodies so that their center of mass lies above their feet. Lee et al. introduced a method to minimize joint stresses in the articulated figure when performing lifting motions [33]. The lifting motion is still kinematic because it is created by solving for a sequence of static poses rather than a continuous dynamic motion.

The system proposed by Perlin in [44] uses raised sines and raised cosines to manipulate joint angles in real time. Noise functions are added to give detail to the motions so that the character appears to shift its weight and reveal a human-like personality. Functions can be blended together using a simple scheme of linear weightings, thereby keeping the system fast and interactive. The raised sine and cosine functions make it possible to achieve smooth transitions between motions. An example of the results that can be achieved through this type of procedural animation can be seen in a video "Danse Interactif" [43], where a woman is shown to dance the rhumba and then do a pirouette.

A kinematic method for animating articulated figure locomotion was developed by Calvert and Bruderlin [10]. Their approach to generating a realistic bipedal walk involves using a cubic spline to animate the hips, and making use of experimentally measured motion data to kinematically position the spline's control points depending on the supporting leg and phase. Several parameters are available for the user to alter the speed and style of walk. There are some similarities between this approach and ours in that splines are positioned over the feet to determine the trajectory of a point mass and then the rest of the body is added around the point mass. However, our method differs because it can be applied to quadrupeds, it takes physics into account, and it also can be used for a greater variety of motions such as running, climbing, jumping and skating.

2.2.3 Motion Capture

The most direct method for generating keyframes for an articulated figure is to record the motion of a real person or animal, and then fit the motion to the character to be animated. The process requires an actor to be fitted with devices which measure the 3D positions (and sometimes orientations) of specific target points. The motion of the target points on the actor are recorded in real-time. A common problem with motion capture is the presence of noise in the data which must be removed through filtering before the information can be used. Although it is often easy to record an animation, multiple attempts are usually required in order to get the desired animated motion.

Current research in motion capture involves building a set of tools which make it possible modify and reuse existing data. To assist the user in modifying recorded data, Bruderlin and Williams developed a motion editor that allows the user to manipulate the data with various signal processing techniques [11]. Among other things, the user can filter motions or blend several of them together.

In some cases, the dimensions of the actor do not match the desired dimensions of the animated character. In this case, the data has to be manipulated so that the joint positions correctly match those of the character and the motions of those joints correspond to motions that would have been made by such a character. Ko and Badler use anthropometric scaling to modify motion data from a person of one size to a computerized human of another size [30].

Motion capture data can be further manipulated by adding signals which alter the style of motion, such as done in [57] by Unuma et al. They employ a Fourier series representation of the data and then compare two different styles of motion to extract the characteristic motion properties which define the style. The data which defines the desired style of motion is added to other motion capture data at various scales to effect *emotion*.

Other work in motion capture addresses the reusability of the data itself. In [67], Witkin and Popovic describe how various motion clips can be blended together. Although the technique is used in computer games, it is very simple and does not handle situations such as self intersection. In [61], van de Panne and Lamouret discuss generating databases of many motion clips, and then extracting the best fitting clips to blend together and generate a complete animation. The authors noted that the resulting animation in their prototype implementation lacked some of the realism in the original motion clips. However, the idea of using large motion databases to extract examples of motions looks like a promising area for those interested in real-time character animation.

2.3 Dynamics

2.3.1 Simulation

Dynamics involves using physics to determine the motion of an articulated figure. It was first introduced in the context of animation by Wilhelms and Barsky [65] as a possible way to animate articulated structures. Later, Armstrong and Green developed an efficient, recursive algorithm to solve the equations of motion for articulated figures with a tree-like topology [3]. A good description of the formulation of the equations of motion for articulated figures can be found in the robotics literature [17].

The main advantage of physical simulation is that it constrains the motion of an articulated figure to be realistic in terms of obeying the laws of physics. The main drawback is that it is problematic to control. It is a difficult task to determine the necessary forces and torques to be applied by simulated muscles in order to perform a specified action. This is in part because the motions of the joints are interdependent. To date, the control problem has limited the use of fully physical models in animation to simple problems.

Given an articulated figure, there are numerous possible methods for tackling the control problem. Approaches vary from general methods, which can be applied to different types of creatures, to controllers tuned for one particular type of creature. The second approach has the advantage of being easier to develop since it applies to a creature of one particular structure only, but this requires a new controller to be created for every type of creature. In [51], Raibert and Hodgins have developed control algorithms for kangaroos, biped robots, and quadruped robots. The user can regulate the running speeds, change gaits, and make the robots do simple path traversal. Further work in this area can be found in [24], [25] and [26]. In all cases, all of the controllers are hard coded for one particular creature, although there is a unifying principle behind their work. To ease the development of controllers, Stewart and Cremer [55] propose the use of a programming language that allows the user define a state machine, add and remove constraints, and alter the controllers. In their prototype, a simulator called Newton was used to process the language and simulate the results.

Usually, finding the correct parameters for a controller requires the use of trial and error on the part of the animator. Some research centers on automating the generate and test method by developing algorithms that can calculate the optimal control parameters for achieving particular motions for various types of creatures. van de Panne et al. use a generate and test procedure to determine a set of poses and time steps that result in optimal locomotion patterns [59]. However, such "virtual toys" may not always be desired because they do not incorporate any knowledge about their environment into their motion. Other work has focused on the use of sensors to trigger muscle responses within the creature. van de Panne et al. connect sensors to the actuator muscles through an intermediate layer of hidden nodes to make a sensor actuator network [58]. An optimization method then determines how the nodes should be weighed in order to achieve an optimal locomotion pattern. Ngo and Marks use a genetic algorithm to choose values for stimulus-response parameters that will generate optimal behaviours according to an optimization metric [40]. Gritz and Hahn use genetic programming to calculate the optimal controllers for a particular motion [22]. Genetic programs, unlike genetic algorithms, can evolve their complexity level. The concept of using a genetic method to generate physically realistic animations was taken further by Sims' work, where optimizations cause the animated creatures to evolve their nervous system and body structure [54]. In this case, the creature's body structure adapts with the controls in order to achieve an optimal form of locomotion. In many cases, if the locomotion pattern is known, it may be possible to guide the optimization of the control parameters. Guided assistance is used by van de Panne and Lamouret to teach a bipedal figure to balance and walk [60]. In a fashion loosely analogous to a child learning how walk, the figure is artificially held upright while it tries to walk. The assistance is then gradually reduced until the walk is learned.

2.3.2 Spacetime Constraints

One way to incorporate physics into an animation while keeping some control over the motion of the articulated figure is to represent the motion using a set of trajectories which define the movement of the degrees of freedom over time. Witkin and Kass first referred to this method as *spacetime constraints* [66]. An optimization functional evaluates a given trajectory-set according to how physically realistic the motion is. The violations of the physical laws are then minimized through optimization. Cohen made several improvements to the method to address the interactivity problems involved with spacetime constraints [13]. His first improvement was to simplify the problem with the use of B-splines to approximate the joint trajectories. This greatly reduced the number of parameters in the optimization. Cohen also introduced spacetime windows so that the optimization procedure is applied to a small subset of control points within a chosen time interval. This simplifies the optimization procedure by making it more local in time. Further work by Cohen and Liu involving spacetime constraints covers the use of simpler models [34], wavelets [36], and speed and time relaxation between keyframes [35].

Currently, spacetime constraint animation requires the use of a symbolic interface to compute the analytic derivative expressions required by the optimization. This complicates the algorithm and has kept it from being utilized in animation systems. Although our work involves the optimization of spline trajectories, it differs from spacetime constraints in several ways. Rather than using trajectory optimization for all degrees of freedom, we use trajectory optimization in a restricted fashion such that it represents the key aspects of the motion. We then use kinematic methods to reconstruct the full motion of the articulated figure. Our underlying motion acts on a highly simplified approximation of our articulated figure: two point masses. As a result, we need to synthesize only two trajectories, one for each point mass. We use a numerical approach which assumes no knowledge of the analytical derivatives typically employed in spacetime optimization methods. This allows for a greater degree of freedom in the choice of terms used in the optimization functional. Finally, we take advantage of the freedom that the physics can be treated as a *soft* constraint. If our footprint constraints make unrealistic demands on the articulated figure, the optimization procedure will *stretch* the physics in order to meet the user constraints.

2.4 Hybrids

There have been several attempts to get the best of both worlds and mix dynamics with kinematics in animations. Spacetime constraints could also be considered a subset of this class because it treats physics as yet another constraint. Since our method is also a hybrid dynamic/kinematic animation scheme, we examine the work in this category in detail.

The KLAW animation system developed by Bruderlin and Calvert [8, 9] uses a hybrid dynamic/kinematic algorithm to animate human walking. Much of their animation model is based on the use of characteristic parameters obtained from human motion analysis. The underlying motion is generated with a simplified 2D dynamics model consisting of two virtual telescopic legs that join together at the hip. Sinusoidal motion added to the hips makes them move over the supporting legs. A human figure is then kinematically constructed over the underlying figure by replacing the telescopic legs with bending ones, and swinging the arms out of phase. Our method also uses virtual telescopic legs and a simple body approximation to represent the underlying motion of the articulated figure. However, the system described in [9] is specific to human walking motions only.

Another approach is taken by van Overveld and Ko [63] who suggest a method to mix physics and kinematics using approximate dynamics methods on point masses. The human figure is approximated by a 14 DOF simplified human body model that is defined by point masses. At interactive rates, the program accepts an instantaneous walking direction and gait length. Internally, it then positions the footprints kinematically, and calculates the trajectories of the foot point masses, and other key bodyparts. The rest of the body is animated with passive dynamics. To get the figure to stand up they invert gravity. The resulting animations have a *cartoony* motion, and the authors admit that this approach will not generate realistic animations of articulated figures. However, the method is fast and the animations exhibit some natural motion due to the inertial effects of the passive dynamics.

Kokkevis et al. use a simple underlying model to animate a dog that the user can control interactively by specifying its desired velocity and heading [31]. The foot positions are controlled kinematically using a kinematic gait controller. At every instant in time, the system calculates the required forces and torques that must be applied to the dog's center of gravity (COG), in order to satisfy the user's velocity and heading specifications. Linear programming is used to distribute the required forces and torques among the legs currently in contact with the ground. If it is not possible to achieve the desired forces and torques on the COG from the current leg configuration, then the distribution resulting in the closest match is found. The resulting forces are then used to move the COG and the torques alter the body's orientation. Afterwards, the inverse kinematics for the legs are added, the tail is animated with a periodic function, and some random motion is added to the head. They manage to generate real-time animations of a dog that can walk or trot over level and slightly inclined ground. However, the resulting motion appears stiff, which they attribute to their simple kinematic model.

Our work is similar in several respects to that of Girard's PODA animation system [18, 19]. In PODA, the user has to first define a set of posture sequences which are essentially keyframe goals that the animal's limbs must achieve fixed instances of time. The user interactively specifies the type of gait and desired body path of the animal. PODA then generates a *support profile* that defines which legs are in contact with the ground at each time instant. The rest of the animation is generated using various rules, which are explained later, to calculate the trajectory of the body. The foot trajectories are calculated using an optimization procedure to minimize jerk, defined as the rate of change of acceleration. Finally the legs are positioned using inverse kinematics. Girard has extended his method to animate a bipedal figure over a set of footprints [20, 48]. Although few details are available on the implementation, we will discuss several known differences.

There are several key differences between our method and Girard's. The most important distinction is that we use an optimization algorithm to calculate the trajectory of the center of mass from the footprint information. The key advantage of the optimization is that all of the rules employed in [18, 19] to translate and rotate the animal's body in a realistic way are implicit in our method. The rules involve the calculation of acceleration that each leg may apply on the body for horizontal movement, the calculation of a banking angle and foothold adjustment on turns, the sinusoidal functions blended together to model varying degrees of knee bending during the support phases, and the calculation of the vertical speeds during the support and non-support intervals. We achieve the outcome of the rules in terms of a compact optimization problem. Aside from lending itself to easy implementation, this allows for correct responses to additional forces not accounted for by Girard's model.

2.5 Miscellaneous

2.5.1 Robotics

Legged motion has been studied in the context of robotics in order to provide control strategies for walking robots. Although robotics research provides us with some insights on the locomotion of legged figures, much of the research centers on slow, statically-stable walking motions. For animation, however, it is not necessary to construct a working controller as long as the final motion looks correct to the human eye.

Rather than giving an extensive overview of this area, we limit our overview to research that relates to our work. A comprehensive overview of legged motion in robotics research can be found in [32]. We have already discussed the various simulations by Raibert and Hodgins [51]. Their control methods were also successfully applied to mechanical hoppers, bipeds and quadrupeds having the same overall construction as specified in the simulations [50]. This work continues to be extended, for example by Nagle [39], who is developing a model and control methods to maneuver a planar hopper over rough terrain, and slippery surfaces. Choi and Song studied the development of six-legged robots that walked slowly over obstacles [12]. The walking motion was generated by ensuring that the center of mass of the robot always projected onto its support polygon.

2.5.2 Biomechanics

Because we have an interest in animating animals, we must have some understanding of how real animals locomote. This area has been explored by Alexander [1] who studied the physics of animal locomotion. As an example of the type of result emerging from motion analysis, we see some justification for using a spring mechanism in the back of the animal and for treating the leg like a *pogo stick*. Alexander also studied how the gaits in animals and associated transitions can be viewed as being optimal solutions in some respects [2]. Studies of human walking are also insightful. Simplified models have been proposed in the book *Gait Analysis* [15], by Craik and Oatis, to further our understanding of how most of the underlying motion arises. Much of the terminology that we use is derived from this book. Other studies, such as the work by Inman et al. [27], explore the motion of the center of mass and joint motion during walking. Many of the rules used by Calvert [8, 9, 10] and Girard [19] for animation synthesis are also described in the work by Inman et al.

2.5.3 Artists

Animators are generally more interested in the visual appeal of the motion rather than how accurate the underlying physics is. Artists who draw animals know how their subjects move and particularly the aspects of motion that are important from a visual perspective. Many of their insights are also a potentially useful source of information. We use the work by Jack Hamm [23] for some information on joint limits, gaits, and simplified skeletal structures. Muybridge presents a now-classical pictorial study on the motion of animals [38]. This reference is invaluable for animators because it contains many stop-motion images of animals in motion.

Chapter 3

Footprint Generation

This chapter presents various methods for generating the footprint data which are the input of our minimization algorithm. These methods include allowing the user to directly control the placement of each footprint, positioning footprints along a curved path specified by the user, optimizing an initial set of footprints into an improved set, automatic placement for random wandering behaviour, and the use of skate tracks for generating skating and sliding movements. We first present some of our assumptions about the footprints and their environment.

3.1 Assumptions

There are several assumptions that we make to simplify the foot placement process. We assume that all footprints are placed using a plan view of the terrain. This places a mild restriction on the movement of our creature. For example, it could not place footprints on the ceiling or walls of a room from an overhead plan view, but it would be easy to extend to multiple views.

Throughout the chapter, we will use the coordinate system shown in Figure 3.1. We define a footprint as a a six-tuple, $< i, x, z, \theta, t_1, t_2 >$, such that:

- i indicates the foot this set belongs to, $i \in \{1, 2, 3, 4\}$,
- x and z indicate the position of the foot, i, on the terrain,
- θ indicates the orientation on the terrain,
- t_1 is the time of contact the foot makes on the ground, and

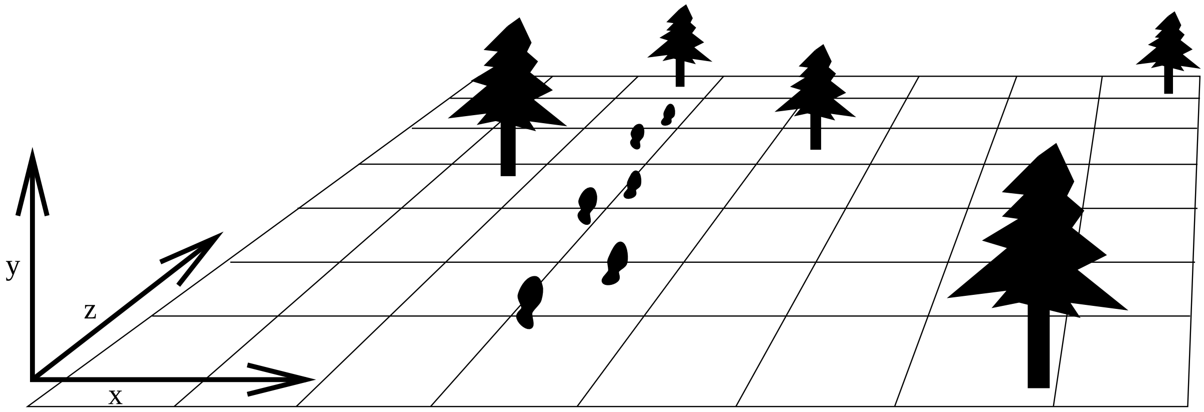


Figure 3.1: Coordinate system for footprint environment.

- t_2 is the time of departure of the foot.

For each footprint, the (x, z) coordinates are determined by the input method and the y coordinate is implicitly specified by the maximum elevation of objects at the given (x, z) location. Because calculating the elevation is frequently required during foot placement and also to determine foot clearance during the generation of the animation, we need an efficient algorithm to determine this. We employ a quadtree structure for this purpose. Each leaf in the quadtree stores an array of pointers to polygons that intersect the leaf's bounding box.

3.2 Direct Positioning of Footprints

The simplest method to generate footprint data is to allow the user to position and orient every footprint using some input device. Although this method can become tedious if the number of footprints is large, it is also the most direct and is useful for fine-tuning an animation. However, specifying the position and timing information of a natural motion can still be difficult for the average user. We have therefore added some features to simplify the process.

3.2.1 User Interface

Our user interface is illustrated in Figure 3.2. The complete environment is seen from a plan view, which the user is free to scale and translate as desired. The objects with

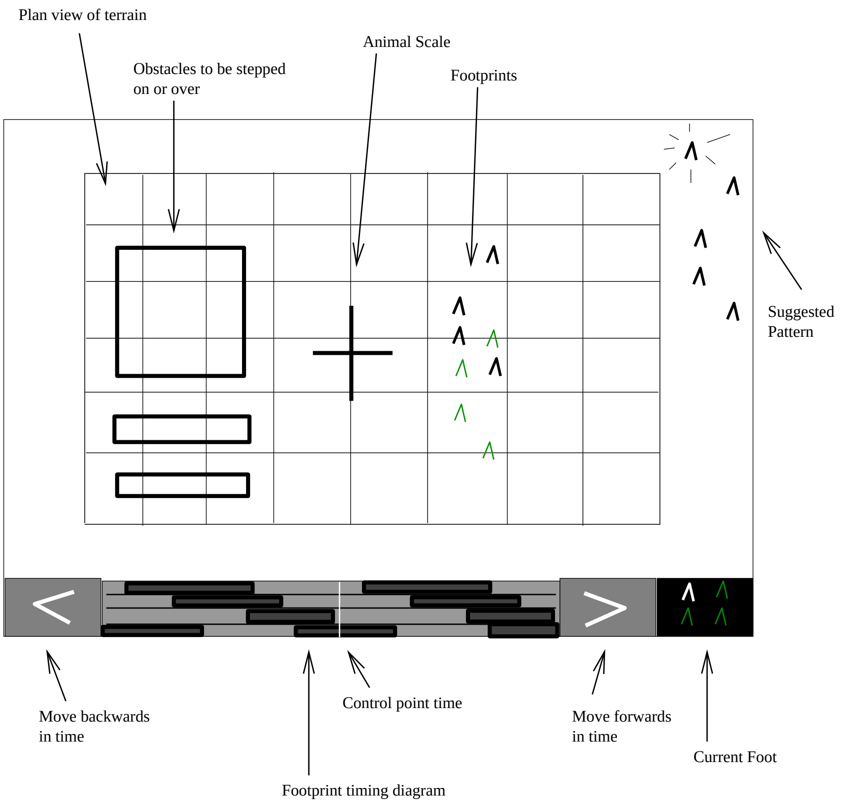


Figure 3.2: User interface for manual footprint entry.

thick outlines represent obstacles in the environment. A footprint is placed by clicking the left mouse button within the terrain grid. By further dragging the mouse before the button is released, the user can specify the orientation of the footprint. Clicking the right mouse button deletes the last footprint placed on the screen. The leg corresponding to the current footprint to be placed is indicated in the bottom right. Screen clutter is a potential problem with using footprints for quadruped animation as the hind footprints may overlap the front ones. Each footprint is colored according to the leg it belongs to, with the last footprint in each set emphasized in bold to further avoid any ambiguity. The user can also make a selection in the timing diagram in order to determine the position of a leg at any time instant.

It is often difficult to make distance judgments without a reference scale. We display a bold cross in the middle of the screen whose dimensions equal the size of the animal's spine. With this, the user can judge how far the legs would stretch at each step. In the top right corner, we show a canonical set of footprints. This is intended to be used as

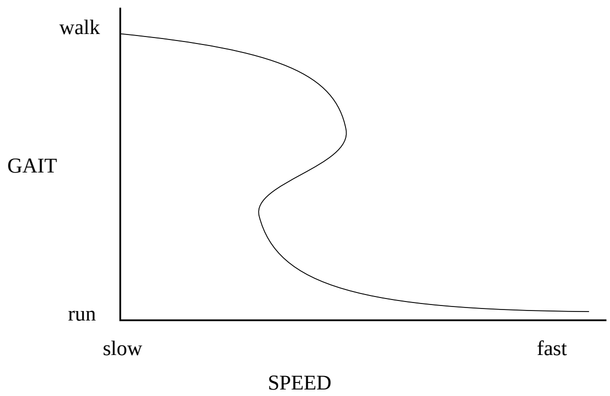


Figure 3.3: Hysteresis commonly exhibited during change of gait.

a further guide for the user. The suggested relative position is highlighted, and the last four footprints appear in bold to allow the user to easily place a new footprint relative to those already positioned.

3.2.2 Timing and Gaits

A new user is unlikely to know the specific timing details of an animal's gait. Even with this knowledge, specifying the timing for every footprint is tedious at best. A better solution is to let the system select the current foot and timing data for each footprint automatically according to a desired gait. For each animal, we have defined three gaits: walk, trot, and gallop. The user is allowed to select any of these gaits while placing the footprints.

Selecting intermediate gaits is possible too, but it requires a little more work than simply interpolating between adjacent gaits. In Figure 3.3 we reproduce a characteristic graph of the gait plotted against the speed of a typical animal [2]. The graph was derived from studies of animals trained to maintain specific gaits at various speeds and illustrates the hysteresis involved in gait selection. Smooth transitions between gaits do not occur; instead there is a sudden change in gait when the speed passes certain thresholds. In [19], Girard shifted gaits by using an optimization procedure that minimized the phase shift between the gaits while enforcing a minimum transfer duration for each leg.

A gait parameter consists of the time to complete one cycle of the gait, and the relative position, time of contact, and time of departure for each foot. We store the gait

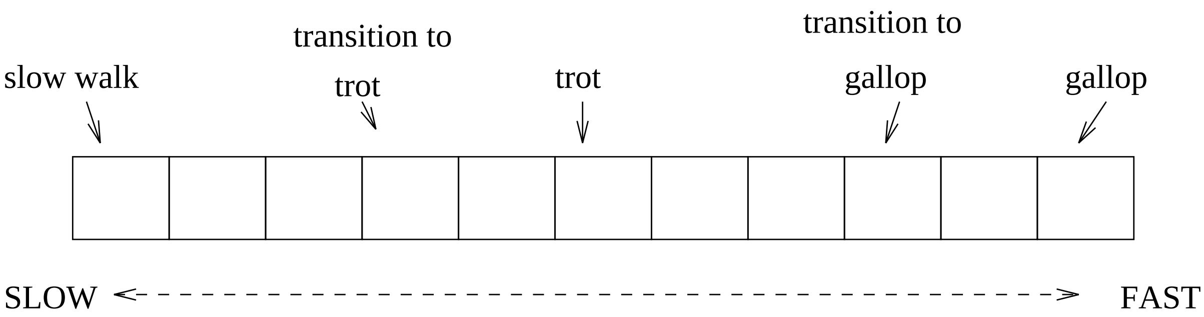


Figure 3.4: Array used for storing gait

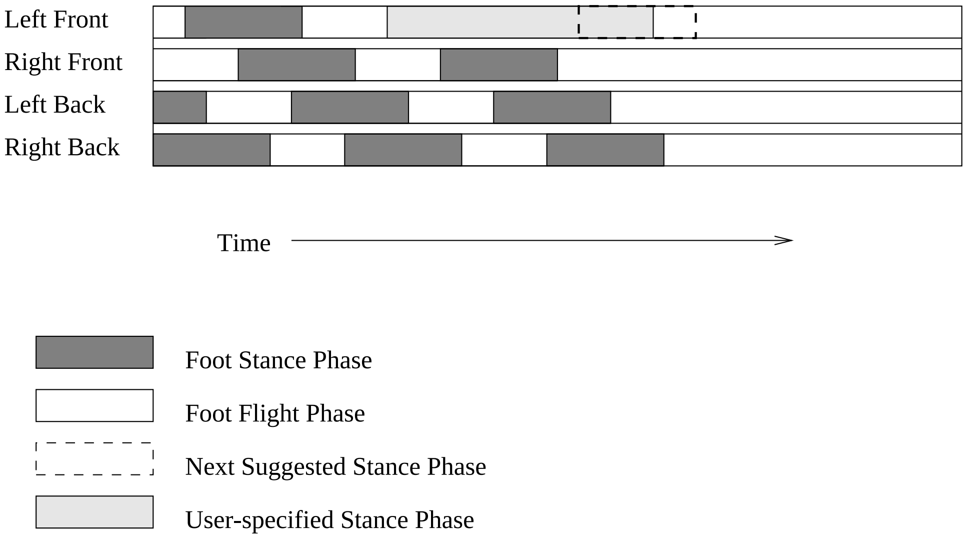


Figure 3.5: Editing of timing information.

parameters in an array, as shown in Figure 3.4. Key gaits can be specified at discrete positions in the array, and the remaining elements in the array are filled by linearly interpolating between adjacent gaits. This offers the user explicit control over how the transitions are performed, although it does not implement the hysteresis. The data for the gaits used in our examples and the various transitions were taken from [21]. The user can select any gait in the table, or instruct for this to be performed automatically. If motions such as jumping are desired, then the timing of the feet may be adjusted by changing the landing and takeoff time of the feet on the time chart with a mouse. The user can alternatively load in a new set of gaits from the database.

If the user concurrently alters the timing information of the footprints while placing additional footprints, it may lead to problems in setting the timing information of the new footprints. For example, in Figure 3.5 we show an edited footprint timing diagram

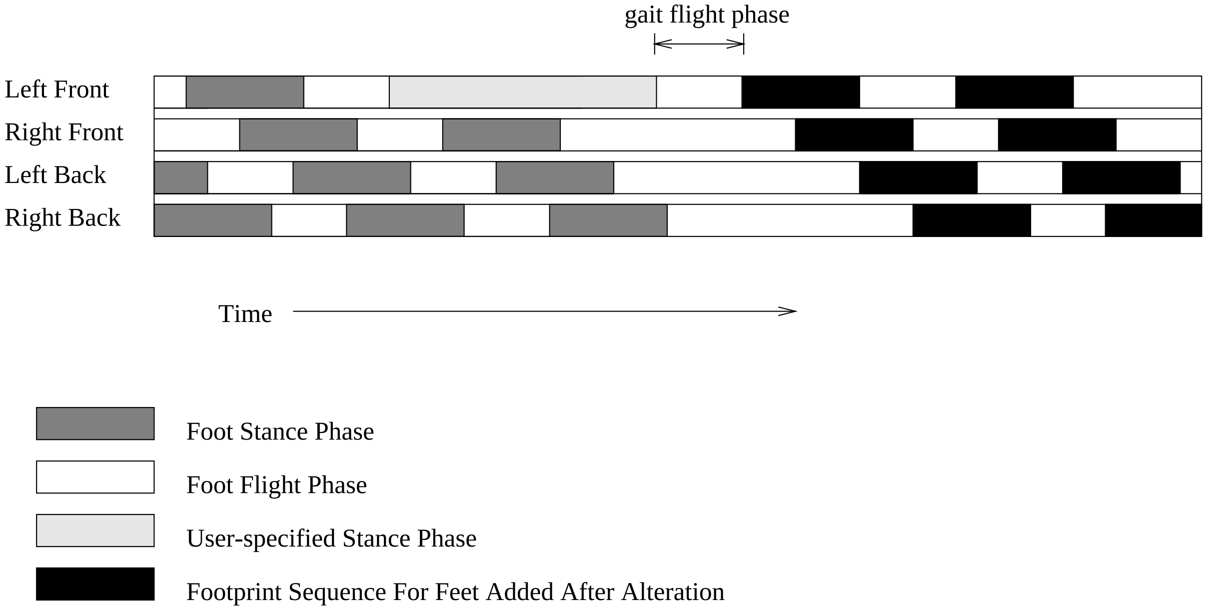


Figure 3.6: Enforcing a minimum flight phase duration for newly created footprints.

where the departure times of the left front leg has been delayed. We cannot blindly follow the gait information for the subsequent footprints, as this could lead to two overlapping stance phases for a foot. How should the times of the next group of footprints be set? We have yet to find a good way to do this, since we do not know how a true quadruped would adjust its gait after extending the contact time of a leg on the ground. For now, we ensure that no overlap in timing information occurs for the newly placed footprints by enforcing a minimum flight phase for each foot, as shown in Figure 3.6.

3.3 Footprints Along a Path

Although it is useful to be able to position footprints one at a time, it can become tedious to do so for many footprints. This tedium can be alleviated with the help of tools that abstract away the footprints. One such abstraction involves using splines to model the path of the animal and to have an automatic process generate a reasonable set of footprints along the path.

It is important that the relative footprint positions are consistent with that expected by a real animal, so we use the gait definition presented in Figure 3.7(a), which includes the relative foot positions, foot orientations, and stride length. Given a position on the

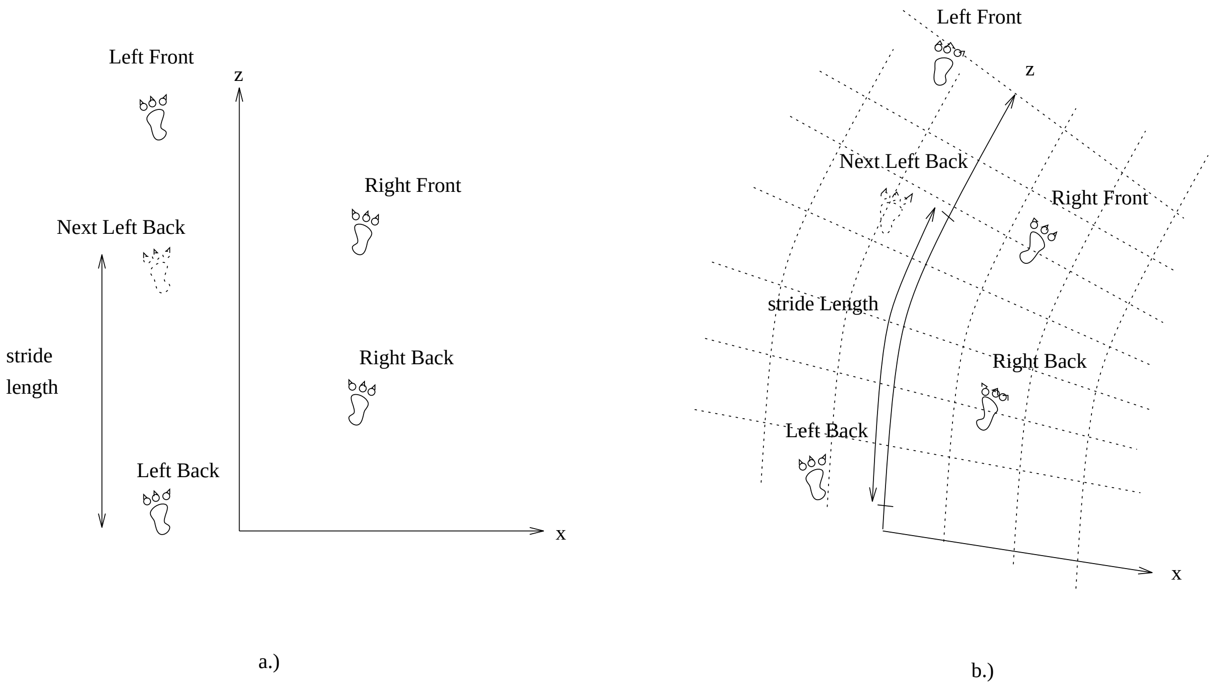


Figure 3.7: Gait parameters. (a) Original gait. (b) Gait warped to fit a curve.

path curve, we calculate the footprint placement by warping the coordinate system so that the z-axis is tangent to the path, and the x-axis is perpendicular to the path on the floor plane, as shown in Figure 3.7(b). We define *stride length* as the linear distance from the footprint position of one foot to the position of the next footprint of the same foot. After the warp, the stride length between footprints may differ from that of the original gait, but the stride length projection on the z-axis is preserved.

Our implementation of the footprint/path positioning algorithm uses cubic Catmull-Rom splines to define the path. An example of the output generated by our algorithm is shown in Figure 3.8.

3.4 Autonomous Motion

Autonomous motion is useful in interactive games and virtual worlds where motion and obstacles are unpredictable. Autonomous motion is also useful when one or more animals are to be animated in the background of a scene. Rather than burdening the animator with the extra work involved in animating background characters that are not the focus of the scene, the characters can be animated autonomously according to a behavioural

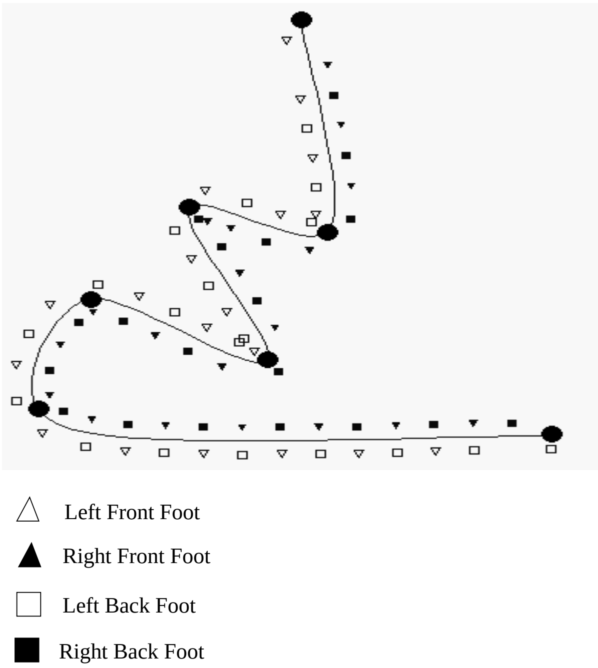


Figure 3.8: Example of footprint generation from a curved path.

model. One possible choice is to first determine various destinations for the animal and then generate the footprints along the path connecting the destinations.

We have developed a version of our animation system that generates a random walk pattern to animate the animal roaming around the terrain. The behavioural model consists of a heading which is randomly altered by a few degrees left or right for each step of the gait. The duration of the gait cycle, d, is set as a function of the change in height of the surface in the direction of the quadruped's body as:

$$d = d_0 + k_1 \arctan(k_2 y), \tag{3.1}$$

where d_0 is the normal walk duration, y is the slope of the terrain in the direction of the quadruped's body, k_1 is a constant that scales the range of durations, and k_2 is a constant that defines how quickly the duration value changes with the slope. We keep the animal roaming within a specific area by manipulating the current angle when the animal walks near the boundaries so that it turns away from the edges.

3.5 Optimization of Footprints

We have briefly experimented with the idea of using our trajectory optimization procedure (to be described in the next chapter) to additionally optimize the footprint positions and times. In such a case, the user specifies a reasonable initial position for the footprints and then the optimization process alters the footprint positions, orientations and timing to achieve a natural motion over the selected terrain. The functional used in the footprint optimization is the result of the trajectory optimization, as described in Chapter 4. Thus we would have a double optimization procedure consisting of an internal function that optimizes the trajectory of the quadruped over the footprints, and an external function that evaluates the quality of the trajectory optimization in order to decide an optimal set of parameters for the footprints themselves. This is more expensive than the trajectory optimization because we are now utilizing the trajectory optimization within an inner loop. The multi-scale heuristic algorithm listed in Figure 3.9 is used to calculate the optimization.

The results of the optimization procedure are mixed. Footprints which are badly misplaced are correctly displaced to a more proper location, but their final location and timing still result in motions which exhibit some awkwardness. Further studies need to be carried out to develop a better fine scale optimization criteria, in order to avoid local minima.

3.6 Sliding Feet

Footprints need not be static. In fact, if we use moving footprints, or *tracks* as our underlying footprint primitive, then the static footprint is merely a special case. Tracks are useful for motions such as sliding or skating. For example, with tracks we can make an animal run, and suddenly stop while the feet skid on the ground. Other examples include running up a slippery hill and simulating motion on a moving object such as a treadmill or escalator. Our tracks are defined by a Hermite curve, as shown in Figure 3.10 for a two legged character. The only additional feature required is the interaction to specify the track for each footprint. For simplicity, the foot is assumed to move along

```
%foot=foot to optimize
%scale=scale of optimization
OptimizeFoot(foot,scale);
energy=optimalTrajectoryEnergy();
flag=1;
while(flag==1)
{
    flag=0;
    for (i=0;i<numFootParameters;i++)
    {
        if (foot.parameter[i].marked=0)
        {
            foot.parameter[i].value+=scale
            temp=optimalTrajectoryEnergy()
            if (temp>=energy)
            {
                foot.parameter[i].value-=2*scale;
                temp=optimalTrajectoryEnergy();
                if (temp>=energy)
                {
                    foot.parameter[i].marked=1;
                }
                else
                {
                    flag=1;
                    energy=temp;
                }
            }
            else
            {
                flag=1;
                energy=temp;
            }
        }
    }
}
```

Figure 3.9: C code for foot placement optimization.

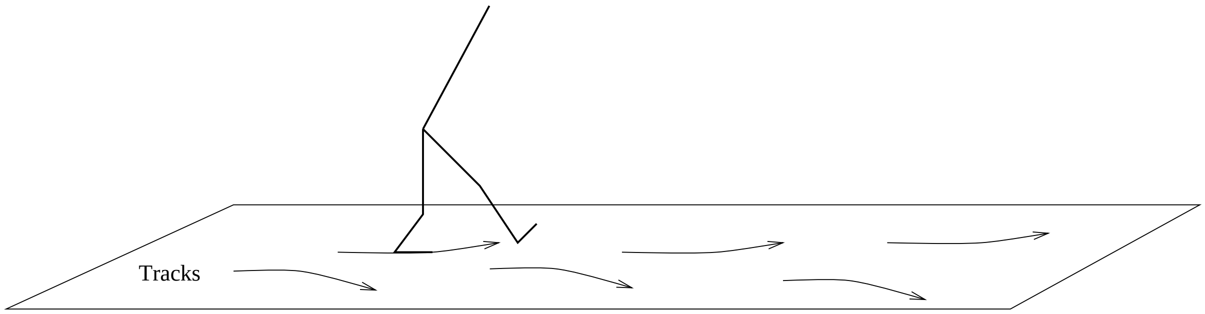


Figure 3.10: Use of tracks to represent foot positions for a bipedal creature.

its track according to the parameterization of the underlying cubic curve. That is, our implementation does not compute an arc-length based reparameterization, which we leave for future work.

3.7 Conclusion

We have looked at several methods of footprint generation and manipulation. Such methods include direct placement, path following, optimization schemes, autonomous motion, and skating. We believe that the field is much richer and more methods will be developed as the utilization of footprints in motion synthesis gains popularity. However, footprints will not develop into a popular animation primitive unless there also exists a method that can use them to generate realistic animations. Our method consists of two phases, trajectory optimization and motion reconstruction. In the next chapter, we will discuss the first phase.

Chapter 4

Trajectory Optimization

This chapter explains the first step in our synthesis algorithm: trajectory optimization. Trajectory optimization allows us to efficiently generate the underlying motion of the quadruped's body over the footprints with the help of a simplified mass model. We reconstruct the rest of the motion afterwards using the methods described in Chapter 5. Our optimization minimizes violations of the laws of physics and ensures that the resulting motion meets the given the current set of footprints constraints. Simultaneously, the motion is also optimized for *comfort* to ensure that the quadruped's body does not assume any awkward positions during the animation. This chapter begins with a discussion of some preliminary ideas that must be understood before we can explain the optimization procedure. Afterwards, we discuss the optimization functional in detail as well as the optimization procedure.

4.1 Preliminaries

This section explains some basic concepts of our trajectory optimization techniques. We use an example to illustrate the main principles behind the optimization, and then discuss how we simplify the structure of our quadruped model and represent the motion of the quadruped's body during the trajectory optimization stage of the motion synthesis.

4.1.1 Overview

An energy function, E, which we define later, is used to quantify the deviation of a trajectory from physics and comfort constraints. We consider an optimal trajectory to be

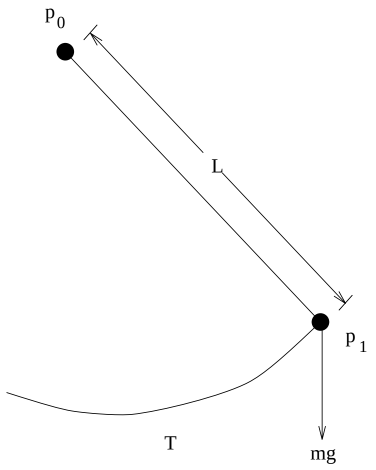


Figure 4.1: Pendulum example.

one that deviates the least from these constraints with respect to E. Deviations from physics are controlled through an energy term which penalizes trajectories with accelerations that cannot be accounted for through our physical model. Our comfort constraints are penalty terms used to maintain kinematic constraints that ensure our model remains visually pleasing. The following example of a swinging pendulum illustrates the general principles behind our approach.

Figure 4.1 shows a pendulum of length L. The point $\mathbf{p_0}$ is fixed, while $\mathbf{p_1}$ is free. For the purposes of demonstration, we will ignore the tension force of the string and maintain the length with a comfort constraint. Since the constraints are *soft*, the string might stretch in order to satisfy our optimization. Because the tension force is ignored, the only force on $\mathbf{p_1}$ is gravity, $m\mathbf{g}$, where m is the mass of the point $\mathbf{p_1}$. T is the trajectory of the point mass, and we use the acceleration of T to determine the total force on $\mathbf{p_1}$. This force, $\mathbf{F}$, is a combination of physically plausible forces, $\mathbf{F_{lp}}$, and physically implausible forces, $\mathbf{F_{ln}}$. That is, $\mathbf{F} = \mathbf{F_{lp}} + \mathbf{F_{ln}}$. Since the physically implausible forces cannot be accounted for, we want to minimize them. From our pendulum model, $\mathbf{F_{lp}} = m\mathbf{g}$, therefore $\mathbf{F_{ln}} = \mathbf{F} - m\mathbf{g}$. So a reasonable energy function for the physics penalty term is:

$$E_{physics} = \int_T k_p \|\mathbf{F}(\mathbf{t}) - m\mathbf{g}\| dt, \tag{4.1}$$

where k_p is a constant to weight the value of the energy functional.

However, while minimizing the physical energy ensures physical plausibility, it does

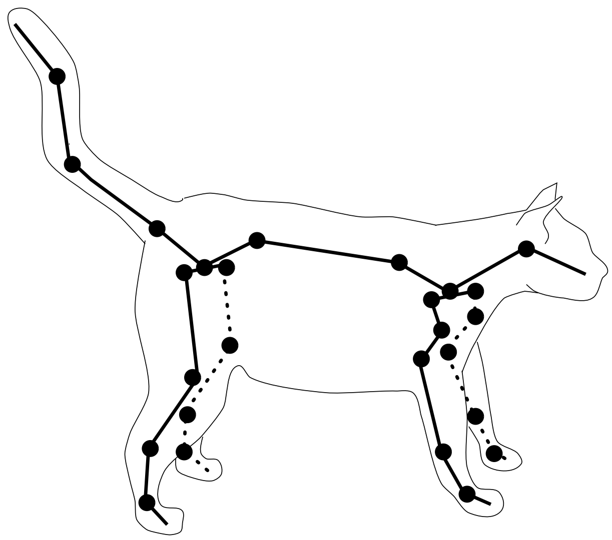


Figure 4.2: Initial model of quadruped skeleton.

not ensure that the animation looks correct to the human eye. In our case, the motion will not look like that of a pendulum at all because we have no energy terms to maintain the distance between points $\mathbf{p_0}$ and $\mathbf{p_1}$. So we add a comfort constraint of

$$E_{comfort} = \int_T k_c(\|\mathbf{p_0} - \mathbf{p_1}\| - L)^2 dt, \tag{4.2}$$

where k_c is a constant to weight the energy functional. So the final functional that we wish to minimize is

$$E_{pendulum} = E_{physics} + E_{comfort}. \tag{4.3}$$

We will then use the trajectory with the minimum energy value in our final animation.

4.1.2 Simplified Body

In Figure 4.2 we show the initial skeletal model of the quadruped to be animated. Because this model has many degrees of freedom, it would be very difficult to control the motion of the model with physical simulation. Our belief is that a full model is not necessary. The initial model can be approximated by a simpler one which is easier to work with, yet can still provide for the general underlying motion of the articulated figure.

If we approximate the body's internal structure with n point masses, $p_1, p_2, \ldots, p_n$, we can represent the underlying motion with n trajectories - one for each point mass, as

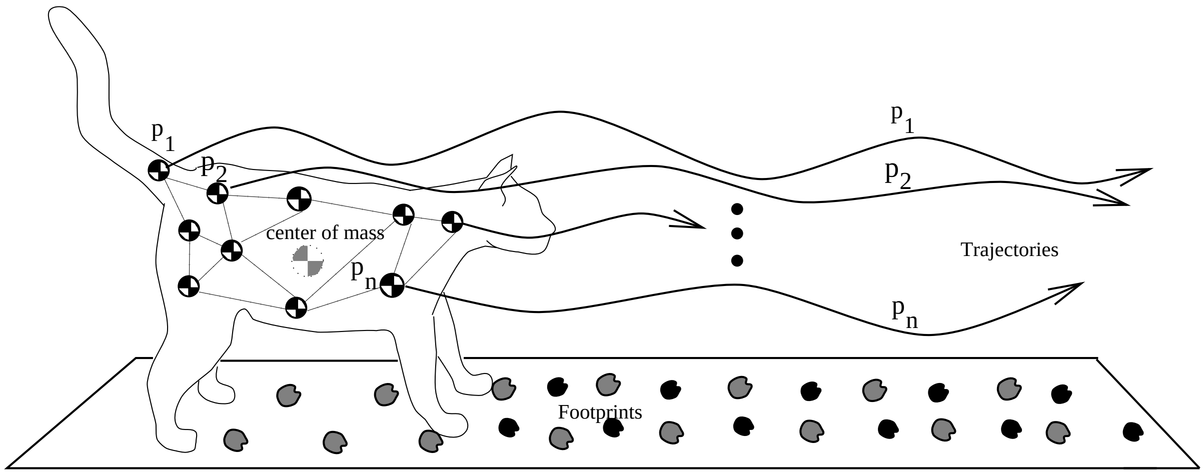


Figure 4.3: Generalized point mass approximation of quadruped's underlying motion.

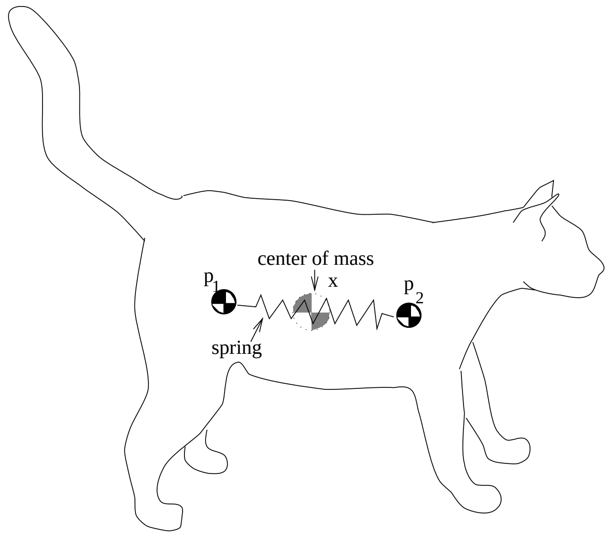


Figure 4.4: A two point approximation of underlying body structure.

shown in Figure 4.3. This breaks up the problem of motion synthesis into two stages: trajectory optimization of the approximate model, followed by full reconstruction of the initial model. As n increases, the physical accuracy of the resulting animation also increases, but at a cost of slower optimization times and more local minima in the optimization. The simplified model used in our prototype implementation consists of two point masses. The internal structure of this model is shown in Figure 4.4.

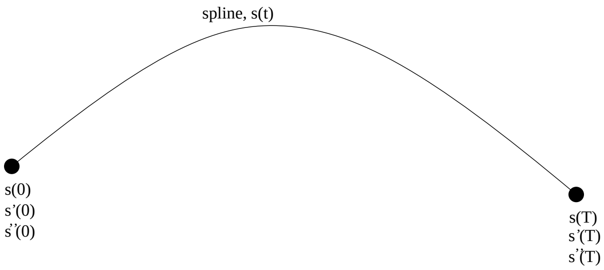


Figure 4.5: Spline used in trajectory representation.

4.1.3 Trajectory Representation

The trajectory of the point masses is represented using spline curves in order to yield smooth motions for the respective masses. We desire that the control points exact local control and that the entire path be C^2 continuous. The continuity constraint reflects the reality that an animal's legs are between the body mass and the environment. We employ a polynomial of degree 5 for our spline and use a Hermite-style definition. Figure 4.5 shows the defining parameters. The derivation for the basis matrix is shown in Appendix A. In general, the requirements of the spline are:

- C^2 continuity
- local control

We do not require interpolation between the control points. The choice of spline depends on the implementor's preference, and our method would also work well with a standard cubic B-spline, for example, to get a smooth trajectory.

4.2 Optimization Functional

The optimization happens over the multiple curve segments used to define the two point mass trajectories. The optimization functional to be minimized is $E(V)$, where

$$E(V) = \int_0^T [E_{physics}(V) + E_{comfort}(V)]\, dt, \tag{4.4}$$

$V = \{V_1, V_2, \ldots, V_n\}$, n is the number of spline segments, V_i is the set of free parameters for curve segment i and T is the duration of the animation. $E_{physics}$ incorporates the

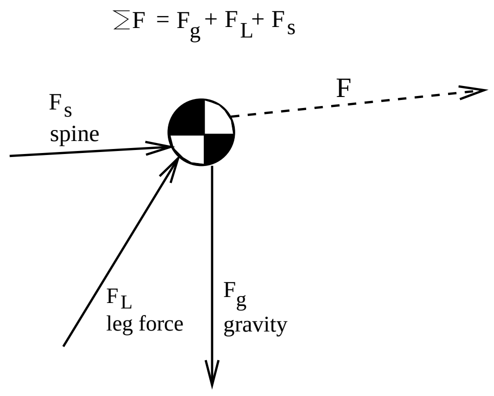


Figure 4.6: Forces involved in point mass physics calculations.

physics constraints. $E_{comfort}$ keeps the animal from contorting into awkward positions by penalizing trajectories which violate certain constraints, which will be discussed shortly.

4.2.1 Physics Constraints

$E_{physics}$ is composed of three terms: one for each of $\mathbf{p_1}$, and $\mathbf{p_2}$, and one for the center of mass. These physical terms attempt to constrain the motion of the point masses such that the applied motions are physically plausible. Figure 4.6 shows a point mass and the forces applied on it. $\mathbf{F_L}$, the leg force, is the only unknown force. The spine is modeled as a spring connecting $\mathbf{p_1}$ and $\mathbf{p_2}$ together, as in Figure 4.4. The spine force, $\mathbf{F_s}$ is applied to the two point masses in equal and opposite directions. It is given by:

$$\mathbf{F_s} = k_s(l - l_o)\frac{(\mathbf{p_2}(t) - \mathbf{p_1}(t))}{\|(\mathbf{p_2}(t) - \mathbf{p_1}(t)\|}, \tag{4.5}$$

where k_s is a spring constant, l is the length of the spring, and l_o is the rest length of the spring. The spring force for $\mathbf{p_2}$ at time t is equal to $\mathbf{F_s}$ but in the opposite direction.

For each of the physical constraints, we calculate an energy function whose magnitude is an indicator of how plausible the force applied by the legs appears to be with the smaller magnitude implying a more "realistic" leg force. This is now discussed in more detail.

Center of Mass Acceleration

To calculate the balance of forces on the body of the quadruped, we need to be able to derive the position of the center of mass and its acceleration from the two trajectories.

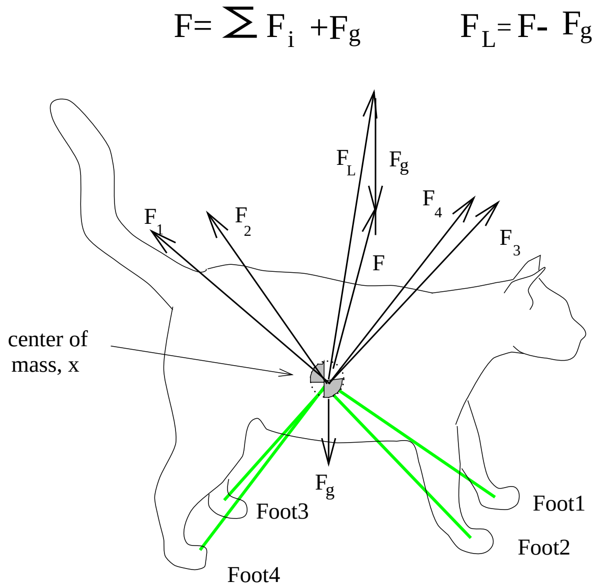


Figure 4.7: Forces applied on center of mass.

Given two point mass trajectories, $\mathbf{p}_1(\mathbf{t})$ and $\mathbf{p}_2(\mathbf{t})$, we can calculate the motion of the center of mass, $\mathbf{x}$, as:

$$\mathbf{x}(t) = \frac{w_1\mathbf{p_1}(t) + w_2\mathbf{p_2}(t)}{w_1 + w_2}. \tag{4.6}$$

where w_1 and w_2 are the relative weights of the point masses, which the user can adjust to alter the position of the center of mass.

$$\frac{d\mathbf{x}(t)}{dt} = \frac{1}{w_1 + w_2}(w_1\frac{d\mathbf{p_1}(t)}{dt} + w_2\frac{d\mathbf{p_2}(t)}{dt}) \tag{4.7}$$

and

$$\frac{d^2\mathbf{x}(t)}{dt^2} = \frac{1}{w_1 + w_2}(w_1\frac{d^2\mathbf{p_1}(t)}{dt^2} + w_2\frac{d^2\mathbf{p_2}(t)}{dt^2}). \tag{4.8}$$

Center of Mass Physics Constraint

Newton's second law of motion, $F{=}ma$, should be satisfied for the center of mass. A free-body diagram of a cat and all of the forces applied to its center of mass is shown in Figure 4.7. $\mathbf{F}$ is the total force, and $\mathbf{F_L}$ is the sum total of all the leg forces applied to the center of mass. For a given pair of trajectories the acceleration, a is known. There is no net effect of the spring force applied to the center of mass, so $\mathbf{F_s} = 0$. Gravity applies a force, $m\mathbf{g}$. Therefore, the force of the legs applied to the body is $\mathbf{F_L} = \mathbf{F} - m\mathbf{g}$. Though we do not have any restrictions on what the magnitude of $\mathbf{F_L}$ should be, our simplified approximation of the quadruped's body allows us to make specific constraints on its direction. We compare the leg force with the region of plausible leg forces allowed by our model, and penalize trajectories that result in implausible leg forces.

The general shape of the region of plausibility depends on the number of legs the animal has on the floor. Since $\mathbf{F_L}$ is the only unknown component of the forces applied to the center of mass, we can assume that the implausible force is in $\mathbf{F_L}$ and that $\mathbf{F_L} = \mathbf{F_{Lp}} + \mathbf{F_{Ln}}$, where $\mathbf{F_{Lp}}$ is the component of the leg force that can be accounted for by our physical model, while $\mathbf{F_{Ln}}$ is the component that is non-physical and cannot be explained by our physical model. We wish to minimize the non-physical leg forces, so our physics penalty is always a function of $||\mathbf{F_{Ln}}||^2$.

For a quadruped, there are 5 different cases that we must treat in order calculate the physics penalty term for a particular leg force. These cases are flight, single support, double support, triple stance and quadruple stance. We will now describe each case in detail.

case 1: flight

A flight situation, as shown in Figure 4.8, implies $\mathbf{F_L} = 0$. In this case, if the leg force derived from our trajectory is nonzero, then we get $\mathbf{F_{Ln}} = \mathbf{F} - \mathbf{F_g}$ for the non-physical component. Our energy functional is then

$$E_{com} = k_{p0}||\mathbf{F} - \mathbf{F_g}||^2, \tag{4.9}$$

where k_{p0} is a constant used to control the magnitude of the penalty.

case 2: single support

If one leg is touching the ground, then the the leg force is constrained to have the same direction as the vector, $\mathbf{L} = \frac{\mathbf{x}-\mathbf{p}}{||\mathbf{x}-\mathbf{p}||}$, connecting the foot to the center of mass, $\mathbf{x}$, as shown in Figure 4.9. The non-physical component of $\mathbf{F_L}$ is shown as $\mathbf{F_{Ln}}$. Note that in

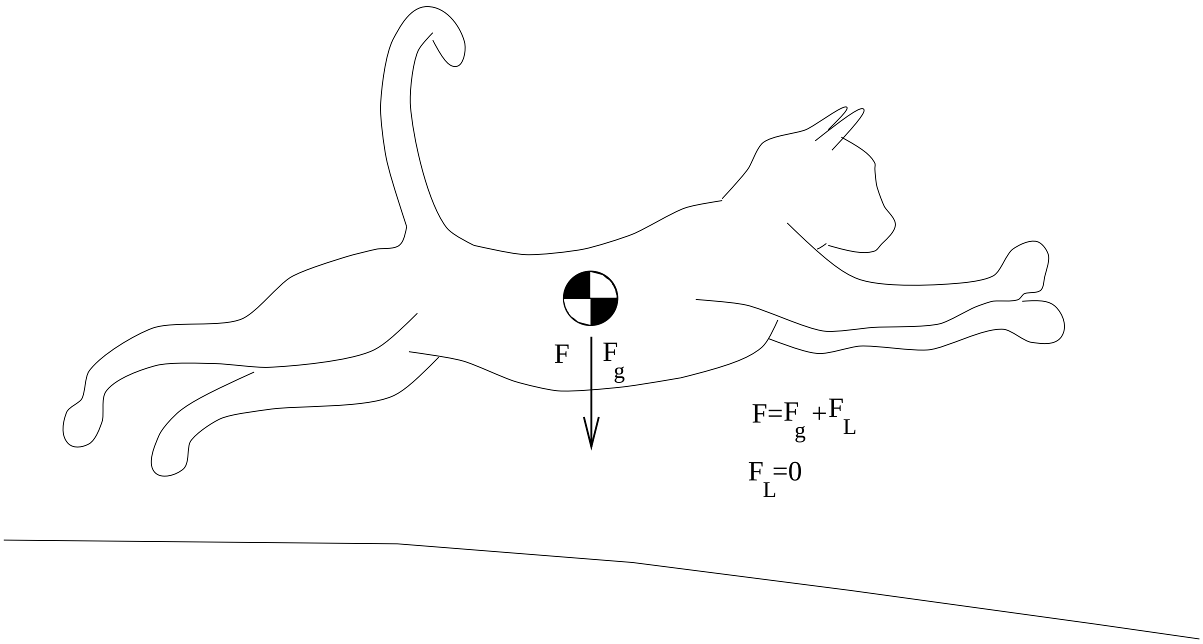


Figure 4.8: Freebody diagram for flight.

Figure 4.9b there are two different non-physical components, $\mathbf{F_{Ln_1}}$ for deviations from the line $\mathbf{L}$ and the other, $\mathbf{F_{Ln_2}}$ for projecting on that line in the opposite direction.

We obtain the penalty for not being parallel to the leg vector by the magnitude of the cross product of $\mathbf{F_L}$ and $\mathbf{L}$. If $\mathbf{F_L}$ projects on $\mathbf{L}$ in the opposite direction, as in Figure 4.9(b), we also add a penalty of magnitude $-\mathbf{F_L} \cdot \mathbf{L}$. Therefore, we set

$$E_{com} = k_{p1}(\|\mathbf{F_L} - \mathbf{F_L} \cdot \mathbf{L}\|^2 + \max(0, -\mathbf{F_L} \cdot \mathbf{L})^2), \tag{4.10}$$

where k_{p1} is a constant used to control the magnitude of the penalty.

case 3: double support

With two legs in contact with the ground the region of plausibility for the leg forces becomes a more interesting. Because each leg can apply a force in the direction from the foot towards the center of mass, the region of plausibility is a wedge shaped region in the same plane as the vectors $\mathbf{L_1}$ and $\mathbf{L_2}$, as shown in Figure 4.10. In this figure, you can clearly see the region is above the center of mass and is bounded by $\mathbf{L_1}$ and $\mathbf{L_2}$. The component of the leg force not in the plane is added to the optimization functional. We also have to penalize trajectories if the leg force does not project within the triangle of

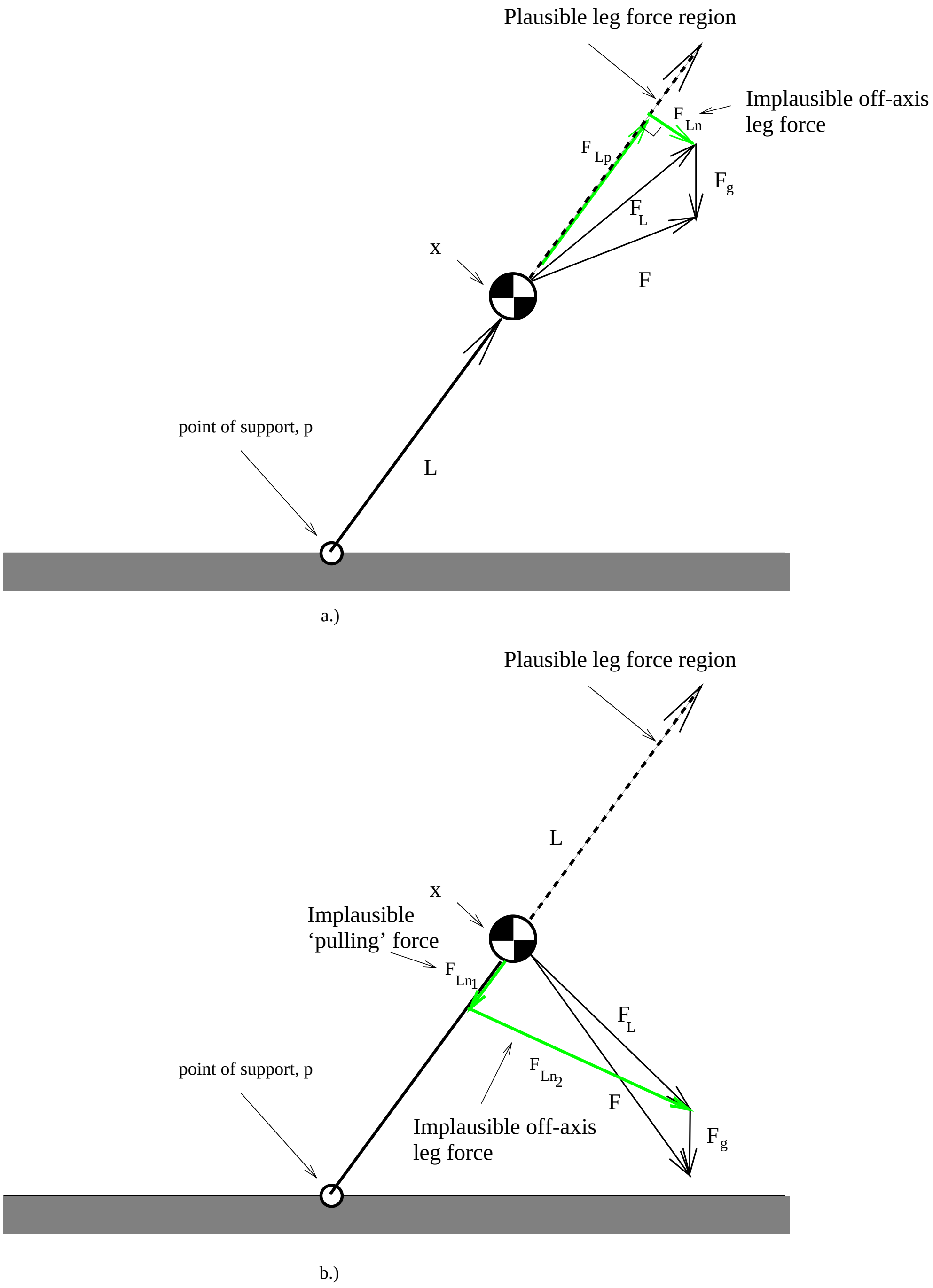


Figure 4.9: Single support.

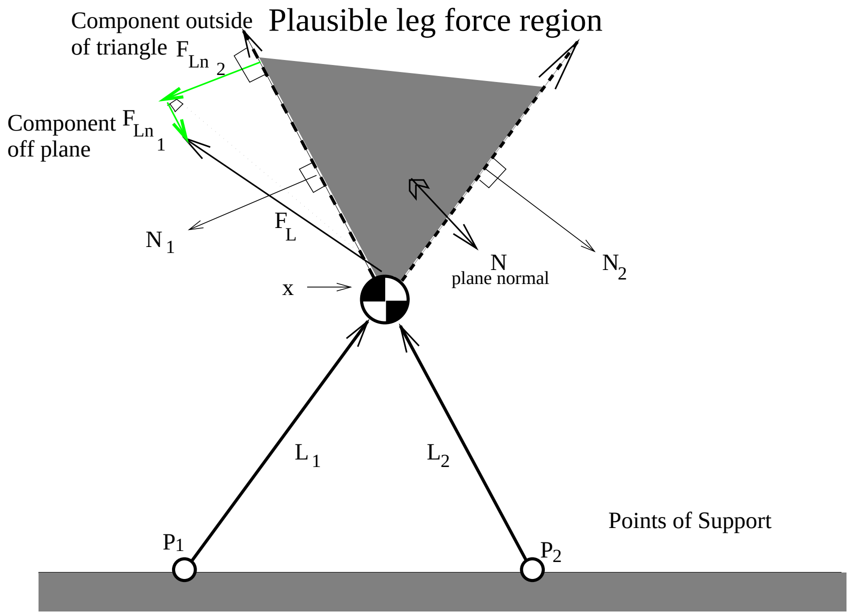


Figure 4.10: Double support.

plausible leg forces. The implausible leg force components are thus penalized directly as follows:

$$E_{com} = k_{p2}(\|\mathbf{F_L} \cdot \mathbf{N}\|^2 + \max(0, \mathbf{F_L} \cdot \mathbf{N_1})^2 + \max(0, \mathbf{F_L} \cdot \mathbf{N_2})^2), \tag{4.11}$$

where $\mathbf{N_i}$ is the vector on the plane normal to $\mathbf{L_i}$ that points outside of the region of plausibility, and $\mathbf{N}$ is a normal to the plane.

case 4: triple stance

When there are three legs in contact with the ground, the region of plausibility for the leg forces forms a pyramid, as shown in Figure 4.11. The support polygon is the triangle defined by the positions of the feet. In a fashion similar to the double support case, the physics term is calculated as

$$E_{com} = k_{p3} \sum_{i=1}^{3} \max(0, \mathbf{F_L} \cdot \mathbf{N_i})^2, \tag{4.12}$$

where $\mathbf{N_i}$ is the normal vector of plane i that points outside of the pyramid. We penalize trajectories with leg forces on the center of mass that do not point inside the pyramid

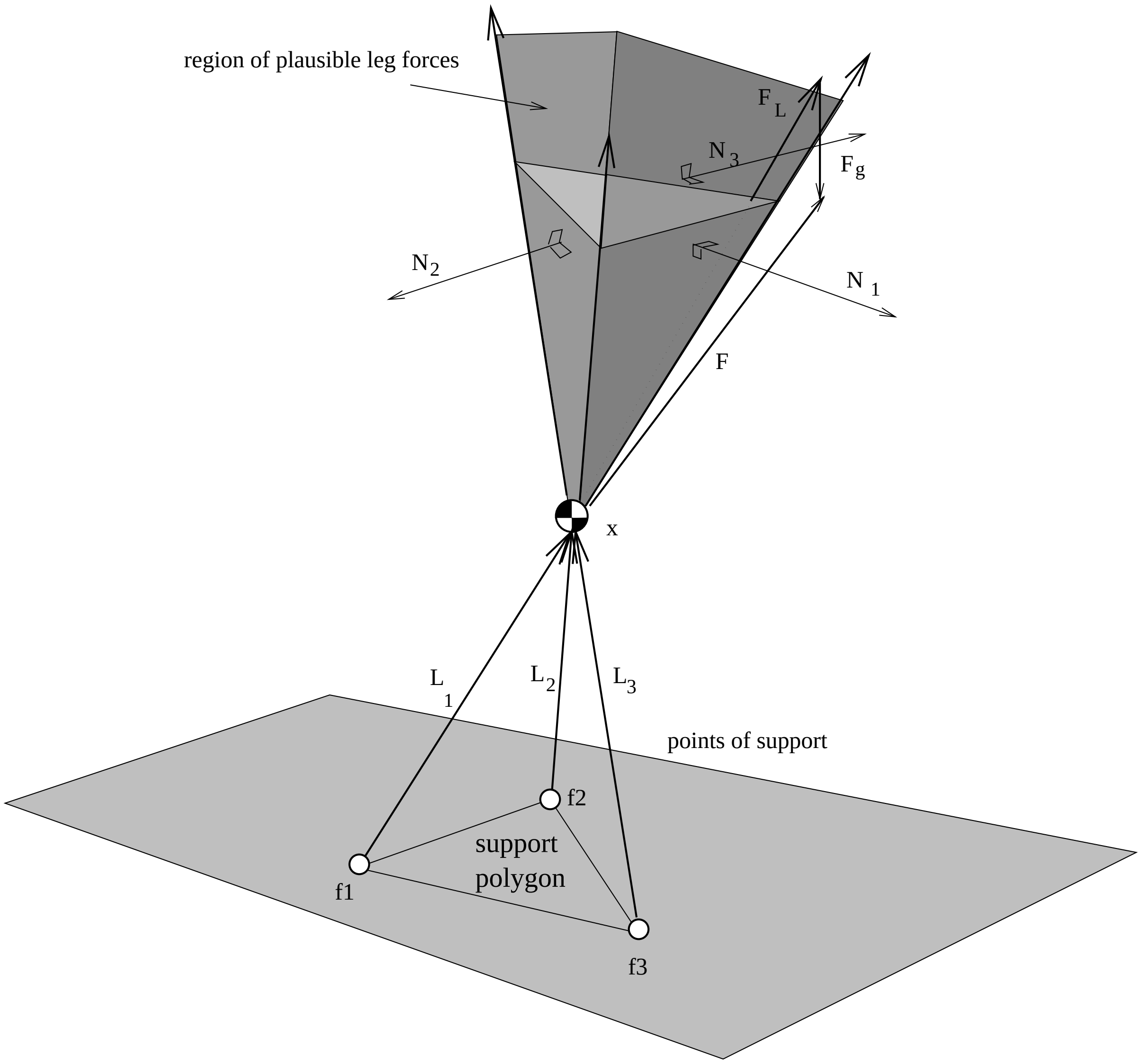


Figure 4.11: Triple stance.

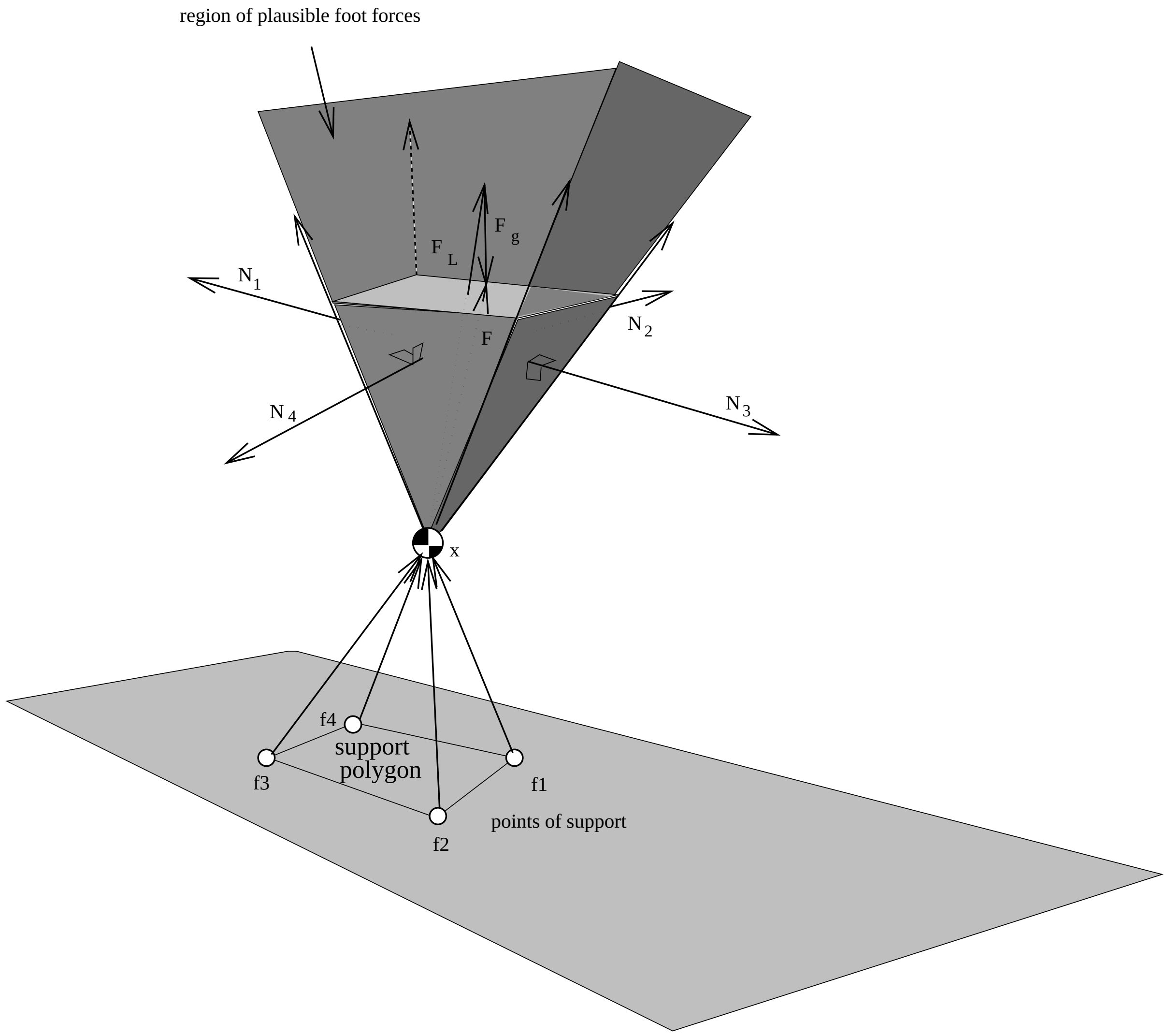


Figure 4.12: Forces applied on center of mass with four feet on the ground.

of plausible forces. We control the magnitude of the penalty in equation 4.12 with the constant k_{p3}.

case 5: quadruple stance

The quadruple stance case is similar to that of the triple stance except that we have a tetrahedron as in Figure 4.12 rather than a three sided one. We compute E_{com} as

$$E_{com} = k_{p4} \sum_{i=1}^{4} \max(0, \mathbf{F_L} \cdot \mathbf{N_i})^2. \tag{4.13}$$

The constant k_{p4} weights the contribution of this term.

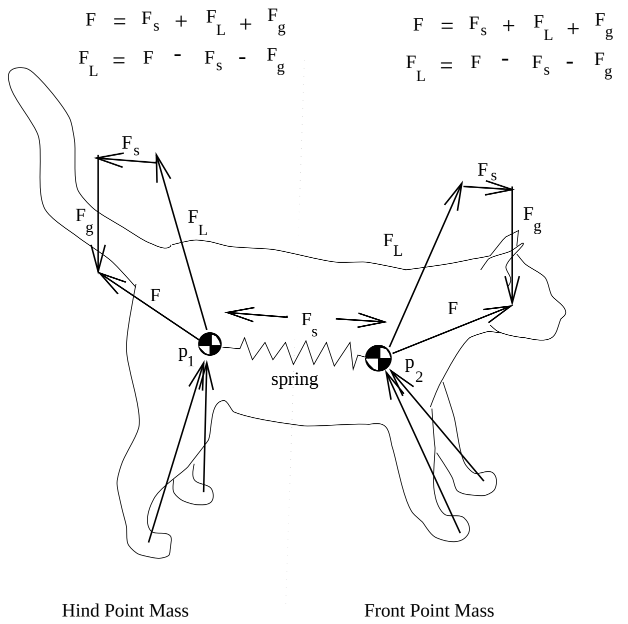


Figure 4.13: Forces applied on front and hind point masses.

Individual Point Mass Physics Constraints

The physical model for the individual point mass physics constraints is shown in Figure 4.13. There is a spring attached between $\mathbf{p_1}$ and $\mathbf{p_2}$ with a force of $\mathbf{F_s}$. Note that this spring force is not 0 as it was for the center of mass. Therefore, we have to include it in the leg force calculations used in this section: $\mathbf{F_L} = \mathbf{F} - \mathbf{F_s} - \mathbf{F_g}$.

The purpose of having individual point mass physics constraints is to localize some physical properties near certain segments of the animal. With only the center of mass physics constraint, it is difficult to properly represent and reconstruct the motion of a flexible back. The point masses would be free to pivot around the center of mass in order to satisfy the comfort constraints to be explained later. On the other hand, with independent physics constraints on the front and back of the body, the animation would look like "two men walking in an animal suit." We therefore allow the user to alter the strengths of the various physics constraints to suit the individual's particular needs.

We calculate the physics constraints for the front and back point masses similarly to the calculations used for the center of mass, except that only one pair of legs is used to provide support: the front or the hind pair respectively. There are three possible cases. We shall refer to the energy functions as E_{pm_i}, where i=1 implies the physics constraint for $\mathbf{p_1}$ and i=2 the constraint for $\mathbf{p_2}$.

case 1: flight phase

As in the center of mass physics constraint case, the force exerted by the legs should be 0. We perform a similar calculation to that of equation 4.9:

$$E_{pm_i} = k_{p0}\|\mathbf{F_L}\|^2. \tag{4.14}$$

case 2: single support

As in equation 4.10 we calculate E_{pm_i} as:

$$E_{pm_i} = k_{p1}(\|\mathbf{F_L} \times \mathbf{L}\|^2 + \max(0, -\mathbf{F_L} \cdot \mathbf{L})^2). \tag{4.15}$$

case 3: double support

As in equation 4.11, we calculate

$$E_{pm_i} = k_{p2}(\|\mathbf{F_L} \cdot \mathbf{N}\|^2 + max(0, \mathbf{F_L} \cdot \mathbf{N_1})^2 + max(0, \mathbf{F_L} \cdot \mathbf{N_2})^2). \tag{4.16}$$

Physics Constraint Summary

The complete physics energy functional is calculated as

$$E_{physics} = K_{com}E_{com} + K_{pm_1}E_{pm_1} + K_{pm_2}E_{pm_2}. \tag{4.17}$$

K_{com}, K_{pm_1}, and K_{pm_2} are weights that the user can use to alter the effectiveness of each constraint.

4.2.2 Comfort Constraints

The physics constraints guide the trajectory optimization to generate physically plausible motions. However, additional constraints are required to ensure that the motion is visually pleasing as well. We apply *comfort* constraints to the trajectory optimization. These comfort constraints are partially motivated by previous work on *strength guided motion* [33]. We use comfort constraints to penalize trajectories that place the animated character in awkward positions. There are many choices for defining which positions

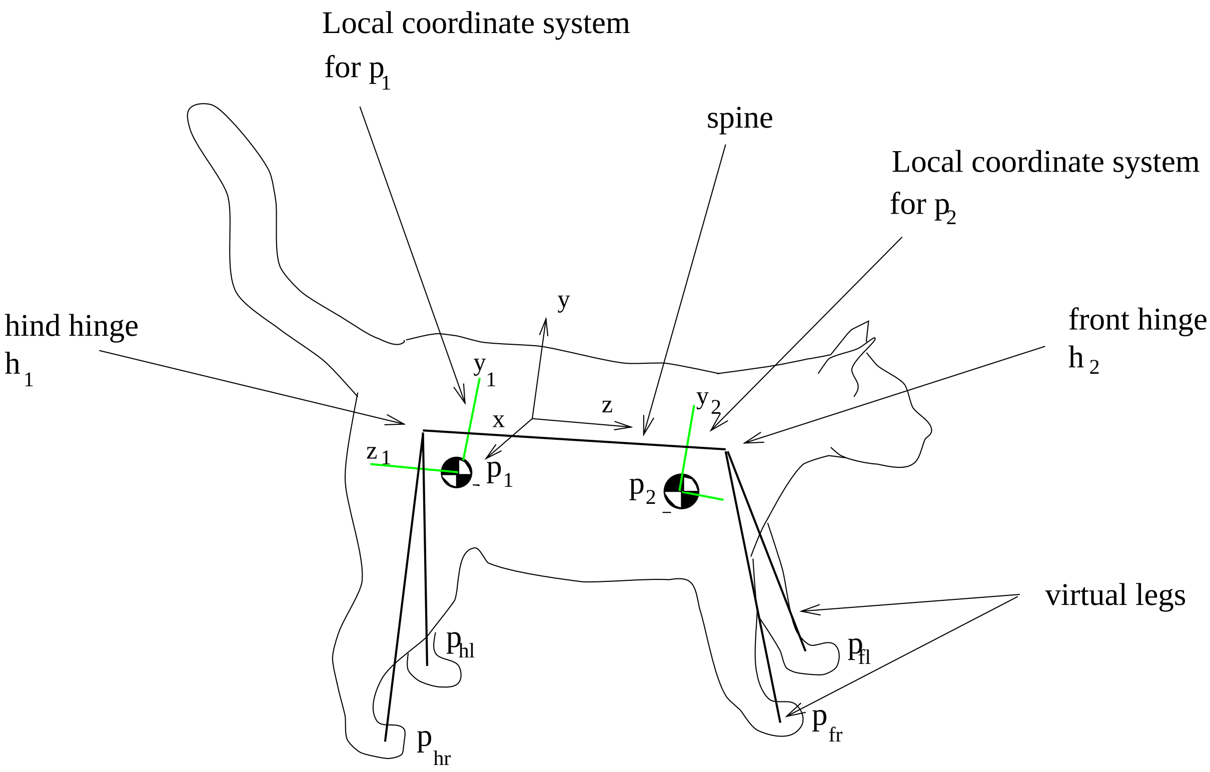


Figure 4.14: Comfort constraint model.

are awkward, and which ones are not. The comfort constraints that we have implemented encourage specific leg lengths, and favours those trajectories where the body of the quadruped does not penetrate the ground. These constraints can be calculated quickly which is an important consideration when doing a numerical optimization which relies on multiple evaluations of the optimization functional. Other possible constraints could involve limits on the angles of the virtual legs with respect to the body, and the shape of the animal at various locations along the trajectory.

Comfort Constraints Model

We build a structure over the two point masses, as shown in Figure 4.14, to evaluate the comfort levels of the animal with respect to its foot positions. Each point mass has a local coordinate system whose z-axis points along the spine and y-axis is perpendicular to the z-axis and points up. The virtual legs attach to the body at the hinges, $\mathbf{h_1}$ and $\mathbf{h_2}$, which are located relative to the local coordinate systems placed on the point masses $\mathbf{p_1}$ and $\mathbf{p_2}$.

Leg Length Comfort Constraints

The comfort constraints on the leg lengths keep the legs from becoming overly compressed or extended throughout the motion. We enforce these constraints by penalizing the trajectory every time a leg touching the floor is farther or closer to the body than the comfort distance allows. The generic equation for this is called E_{lc}, and we calculate it as

$$E_{lc}(p_f, h, l) = \begin{cases} K_{lc}(\|p_f - h\| - l)^2 & \text{if } p_f \text{ touches the ground,} \\ 0 & \text{otherwise,} \end{cases} \tag{4.18}$$

where $\mathbf{p_f}$ is the foot position, K_{lc} is a constant used to weigh the comfort constraint, and $\mathbf{h}$ is the hinge the foot is to remain close to. There is one leg comfort constraint for each leg. The length of the hind legs is calculated relative to $\mathbf{h_1}$ and the length of the front legs is calculated relative to $\mathbf{h_2}$. Each leg has its own comfort distance to allow for the creation of various animation effects like limping or lopsided walking. The complete leg comfort calculation that encompasses all of the leg length constraints is

$$E_{leg} = E_{lc}(p_{hl}, h_1, l_{hl}) + E_{lc}(p_{hr}, h_1, l_{hr}) + E_{lc}(p_{fl}, h_2, l_{fl}) + E_{lc}(p_{fr}, h_2, l_{fr}), \tag{4.19}$$

where the subscript h indicates a hind leg, f indicates a front leg, l indicates a left leg, and r indicates a right leg.

Body Collision Constraint

In most of the animations we generate, E_{leg} sufficiently shapes the motion to keep the animal from contorting into awkward positions. However, during ambitious ballistic motions the body of the articulated figure can penetrate the ground below it. We therefore provide for an additional term to discourage this behaviour. As shown in Figure 4.15, for each point mass, $\mathbf{p_i}$, we hang a sensor, $\mathbf{s_i}$=$(p_{i_x}, p_{i_y} - r_i, p_{i_z})$. When $\mathbf{s_i}$ penetrates an object below it with a height of y, we add a penalty term, E_{body_i}, to the optimization functional such that

$$E_{body_i} = \begin{cases} K_{body}(p_{i_y} - r_i - y)^2 & \text{if } p_{i_y} - r_i < y \\ 0 & \text{otherwise.} \end{cases} \tag{4.20}$$

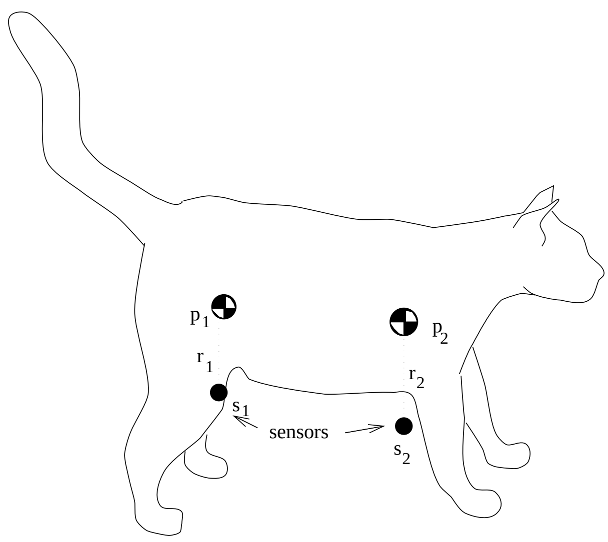


Figure 4.15: Forces applied on front and hind point masses.

The complete body collision optimization functional is then

$$E_{body} = E_{body_1} + E_{body_2}. \tag{4.21}$$

The combined comfort terms give:

$$E_{comfort} = E_{body} + E_{leg} \tag{4.22}$$

4.2.3 Evaluation

We evaluate the integral of equation 4.4 using a summation of the optimization functional at discrete points throughout the trajectory and add up the results:

$$E(V) \approx \sum_{i=0}^{n-1} [E_{physics}(V) + E_{comfort}(V)], \tag{4.23}$$

such that n is the number of spline segments and $V = \{V_1, V_2, \ldots, V_n\}$, where V_i is the set of free parameters for curve segment i. In our model, we use 10 evenly spaced sample points for each spline segment. In the future, we will implement an adaptive quadrature scheme to efficiently evaluate the functional.

4.3 Optimization Procedure

4.3.1 Initialization

Before we begin the trajectory optimization, we must determine an initial trajectory for each of the two point masses. The optimization routine modifies the initial trajectories in order to achieve a pair which minimizes the energy functional. The initialization routine is detailed in Appendix B. In this section we give a brief summary. The trajectories need to be initialized to approximately follow the footprints. Using one spline segment per gait cycle achieves good results. If more spline segments are used, the optimizations proceed slowly and are also more prone to getting stuck in local minima. On the other hand, if we use fewer spline segments then the motion lacks the flexibility to represent lively movements and becomes overly smooth. We initialize the first and second derivatives of the end points of the spline segments to 0. For each control parameter, we estimate the position of the center of mass and orientation of the quadruped at that discrete instant in time. Then we initialize the position of the front and hind point masses to where we would expect them to be in the approximated quadruped body.

4.3.2 Optimization

A greedy, deterministic algorithm is used to optimize the trajectories. The optimization algorithm alters trajectory parameters one at a time, followed by a reevaluation of the trajectory energy function. The set of parameters that defines the lowest energy trajectory is kept and used in the next iteration of our optimization procedure. We take advantage of the local parameterization that our splines provide by limiting the evaluation of the functional to the locally affected areas.

There are two key parts in our optimization, a procedure called **OptimizeNode**, listed in Figure 4.16, that optimizes a given node in the spline at a preselected scale, and a loop that selects which nodes to optimize and decides when the scale of optimization should be reduced.

The outer loop of the optimization, listed in Figure 4.17, processes every node in order and for each node it tries to alter at most one parameter by calling **OptimizeNode**. We

```
%node=node whose parameters are two be optimized
%T1=trajectory of point mass p1
%T2=trajectory of point mass p2
%scale=scale of optimization
function OptimizeNode(node,T1,T2,scale)
    energy=evaluateNodeEnergy(T1,T2);
    loop
        exit when all parameters marked on every axis;
        choose axis  % from (x,y, or z);
        choose parameter % from T(position,
                                  % velocity, or acceleration);

        if (parameter.axis not marked)  %allowed to move along axis
            parameter+=axis*scale;
            temp=evaluateNodeEnergy(T1,T2);

            if (temp>=energy)  %then evaluate in opposite direction
                parameter-=2*axis*scale;
                temp=evaluateNodeEnergy(T1,T2);

                if (temp>=energy) %then could not lower energy along axis
                    parameter+=axis*scale;
                    mark parameter.axis;
                else
                    energy=temp;
                    return 1;
                end if
            else
                energy=temp;
                return 1;
            end if
        end if
    end loop
    return 0
end OptimizeNode
```

Figure 4.16: Pseudocode to optimize one node

```
%optimization outer loop
%T1=Trajectory of point mass p1
%T2=Trajectory of point mass p2

proc Optimize(T1,T2)
    scale=4
    initializeControlPoints()

    loop
       exit when scale<0.25
       numIterations=0;

       loop
           exit when numIterations>100
           flag=0;

           for i=1;i<numControlPoints-1;i++
               flag=flag+OptimizeNode(T1.node(i),T1,T2,scale)
               flag=flag+OptimizeNode(T2.node(i),T1,T2,scale)
           end for
           if (flag==0)
               break;
           end if
           numIterations++;
       end loop
       scale=scale/2
    end loop
end Optimize
```

Figure 4.17: Outer loop of optimization

use a multi-scale approach for efficiency. When the trajectory can no longer be altered at a particular scale, or the maximum number of iterations has been reached, we cut the magnitude of the scale in half and process the nodes again. The routine exits when the scale is smaller than a certain threshold.

4.3.3 Efficiency

Our numerical optimization procedure uses a naive, greedy approach for simplicity. However, we are concerned with the time efficiency of the algorithm as well, so we have made the algorithm multi-scale. During the early part of the algorithm's execution, the perturbations applied to the trajectory are set to be quite large. If the procedure finds that it cannot make further changes at a particular scale, it reduces the size of the scale and repeats the optimization process. The improvement in efficiency comes from the routine applying large perturbations early on in the optimization when the trajectories deviate the most from their optimal state, and making small changes late in the optimization when the trajectory is near its (locally) optimal state.

The other speed improvements are achieved by minimizing the number of calls to the evaluation function of the trajectory's energy. In **OptimizeNode** we mark parameters that did not assist in the minimization of the energy function. This speeds up the routine because if a parameter is not successfully altered in one iteration, then it is likely that it cannot be changed during the next iteration at that scale either. If it could be changed, then it will be altered when the scale is decreased. For the same reason, we mark nodes whose position has not been altered. In this case, we save even more by not having to check every parameter required by this node. Finally, we include a limit on the number of iterations throughout the code to bound the total time of execution to reasonable levels.

4.4 Conclusions

The trajectory optimization allows us to calculate the underlying motion of the quadruped. By using a simplified structure for the quadrupled, we are able to quickly determine to what degree a given trajectory is physically plausible. This is done by comparing the direction of the unknown force, the leg force, with the range of allowable leg forces determined by the position of the feet. Trajectories which require physically implausible leg forces are penalized by our optimization process. Because we have a range of possible physically plausible motions, we can guide the motions with comfort terms which penalize trajectories that place the quadrupled in awkward positions. Now to complete the animation, we have to reconstruct the true skeleton of the articulated figure over the optimized trajectories. We will describe our reconstruction method in the next chapter.

Chapter 5

Motion Reconstruction

In this chapter we discuss the reconstruction of the animal's motion from the point mass trajectories synthesized by the algorithms described in the last chapter. We suggest using a hybrid application of several kinematic, dynamic, and optimization techniques and applying them where they would be most advantageous. Generally, there are several possible choices for each component of the reconstruction. The choice of method will depend on the animator's preference, the quality of animation desired, and the desired frame rate. We shall review the various techniques that can be used to reconstruct the motion and suggest where they are best applied. We also explain the choice of methods used in our implementation to show that a satisfactory reconstruction (subjectively speaking) is indeed possible.

As an example of a reconstruction procedure, Figure 5.1 shows the particular reconstruction scheme we use in our system. The numbers denote the ordering of the various reconstruction steps. A dependency graph describing the relationship between the steps is shown in the upper right corner of the figure. This model is only one example of how a detailed animal motion can be reconstructed from the center-of-mass (COM) trajectories and footprint information.

5.1 Optimization

Given the optimization algorithm used to achieve the COM trajectories, one can further consider optimizing the trajectory of all body parts. With the right optimization functionals, this would potentially provide a very good balance between physically real-

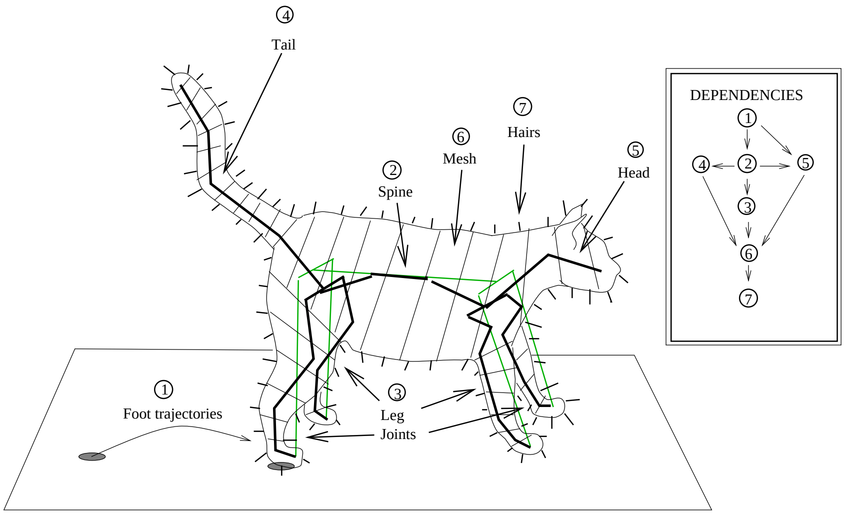


Figure 5.1: Steps involved in the reconstruction of the body. The numbers indicate the sequence of steps. The dependencies are shown in the upper right corner.

istic and visually pleasing motions. However, optimization has the disadvantage of being slower than the other methods to be described later, as well as being prone to producing unpredictable results because of local minima. Therefore, we suggest using optimization to determine the motion-trajectories of only a few body parts. The motion of the feet during their flight phase is a good example. Such motion is usually influenced by the current behaviour, comfort constraints, and inertial effects of the physics. As a second example, a trajectory optimization could be performed to generate the motion of the head such that the creature could exhibit various behaviours such as scanning, sniffing, and tracking an object. At the same time, some physical influence could be added to make the head bounce slightly as the creature is walking. Finally, comfort constraints could be used to keep the head in reasonable positions and orientations with respect to the body.

5.1.1 Optimization for Foot Flight Trajectories

We use trajectory optimization to determine the motion of the feet during their flight phase because we want the feet to move smoothly, to remain near the body, and to avoid obstacles.

Our underlying skeleton for reconstruction, as shown in Figure 5.2, is slightly more complex than the model introduced in Chapter 4, with the notable addition of shoulders

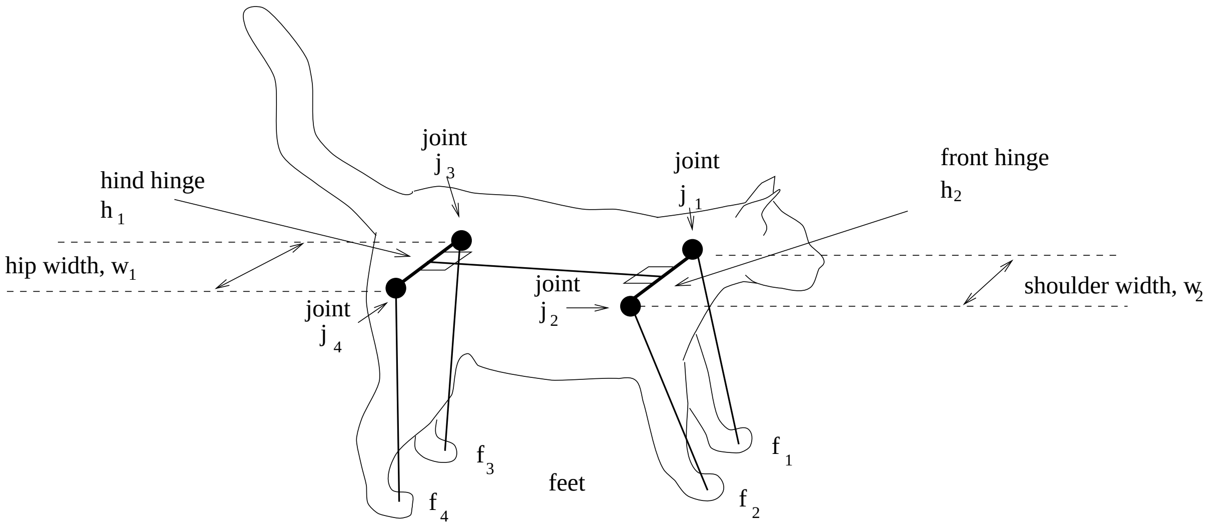


Figure 5.2: Underlying model used in calculation of foot trajectories

and hips, which are aligned using the x axis of the corresponding hinges on which they are placed on. In this figure, $\mathbf{f_i}$ is the position of foot i and $\mathbf{j_i}$ is the associated hip or shoulder joint.

A cubic Hermite curve is used to represent the path of the foot's tip (i.e., toes) as the leg swings in the air. The start and end positions of the curve are fixed by the positions of the two footprints which bound the flight phase, as shown in Figure 5.3. The tangent vectors represent the free parameters in the optimization. A linear interpolation determines the orientation of the foot in mid-flight.

The optimization functional must reflect several goals. First, the foot should move without interpenetrating obstacles, thus giving rise to a *collision constraint*, E_{fcoll}. Also, the foot, $\mathbf{f_i}$ should remain at a reasonable distance from the joint $\mathbf{j_i}$. This is our comfort constraint, E_{fcom}. Finally, we incorporate a notion of physics through E_{fphys} which reflects physics-based energy constraints. The optimization functional to be minimized, E_{flight}, is expressed as

$$E_{flight}(H) = \int_{t_0}^{t_1} [E_{fcoll}(H(t)) + E_{fcom}(H(t)) + E_{fphys}(H(t))]dt, \tag{5.1}$$

where H represents the Hermite curve whose parameters will be optimized, and t_0 and t_1 are the times bounding the flight phase.

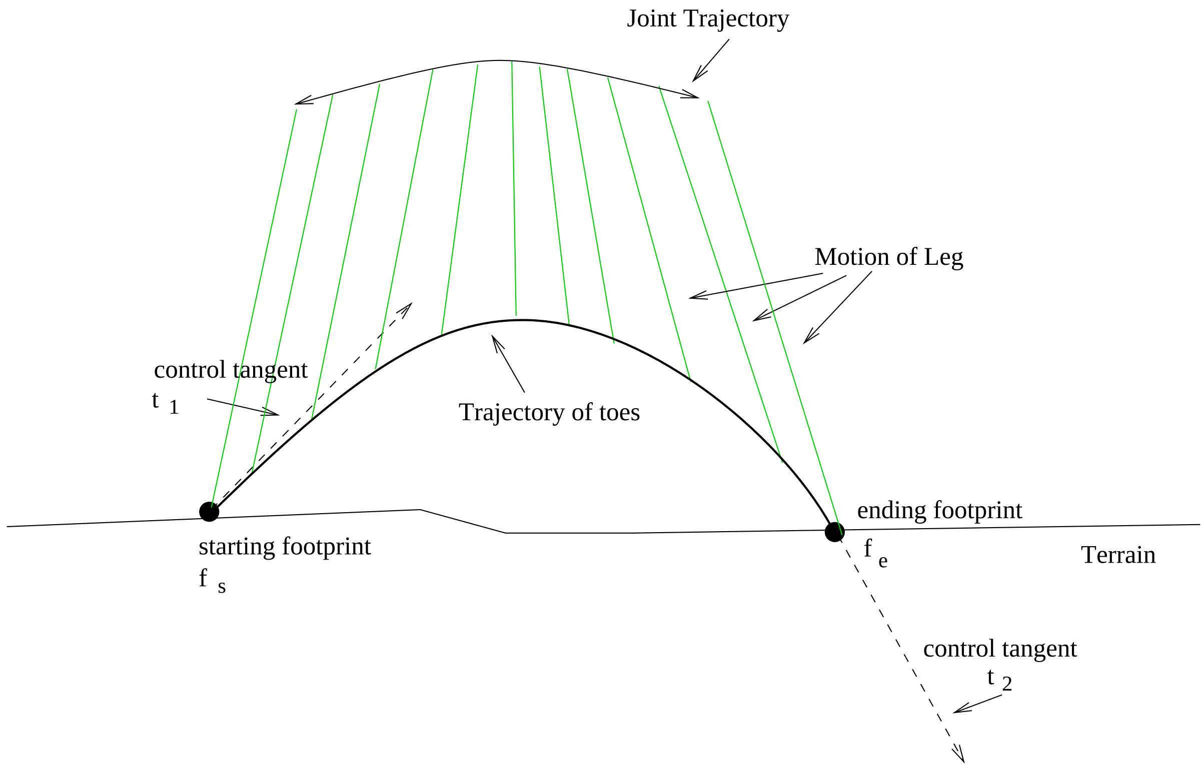


Figure 5.3: Trajectory of the toes during the foot's flight phase, represented by a cubic Hermite curve.

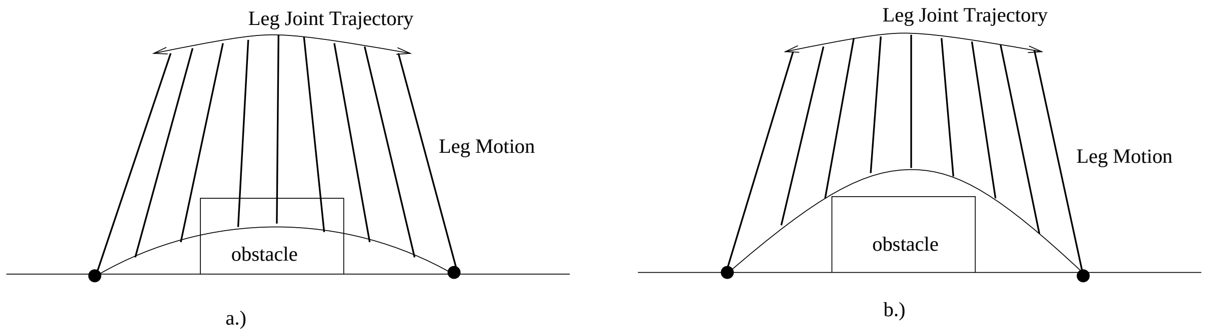


Figure 5.4: Collision avoidance examples for a leg. (a) no collision avoidance, (b) with collision avoidance.

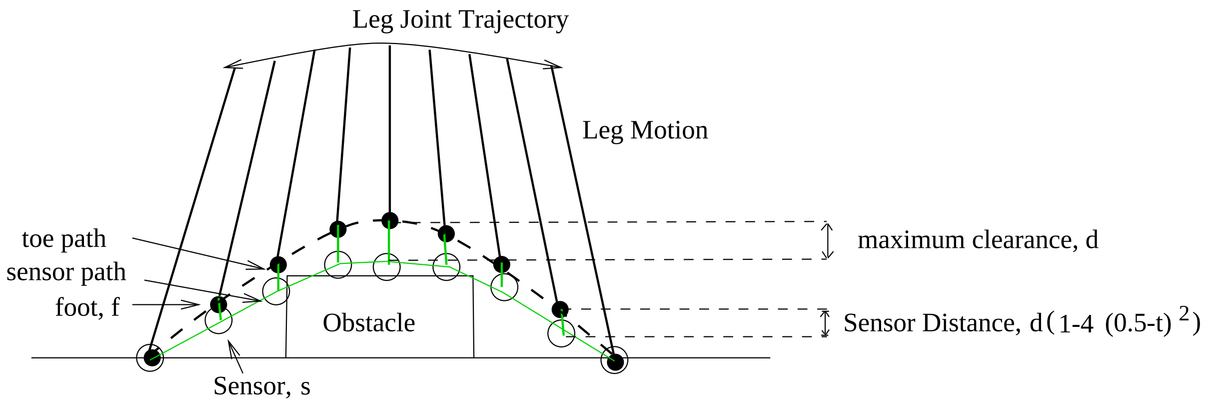


Figure 5.5: Collision detection.

Collision Constraints

Collision avoidance is an important aspect of realistic foot motion. Without collision avoidance, situations such as that in Figure 5.4(a) are common on variable terrains. A more desirable motion is shown in Figure 5.4(b), where the foot passes over or around the object. The collision avoidance should also be efficient because of the potentially large number of obstacles that may be present in a complex environment. Therefore, we limit the collision constraints to ensure that the toes do not penetrate any obstacles on the ground, or the ground itself. As in the simplified collision constraints done for the point mass trajectories in section 4.2.2, a sensor, $\mathbf{s}$, is suspended below the optimization point, which in this case is the foot, $\mathbf{f}$, as shown in Figure 5.5. The distance of $\mathbf{s}$ from $\mathbf{f}$ is described by a quadratic, varying from 0 at the end points of the spline to a maximum distance of d at the mid point of the spline. A quadratic is used to scale the clearance to zero at the end points in order to avoid unnecessarily high trajectories. The clearance term causes the foot to pass over the obstacle with some clearance, rather than brushing close to its surface. This is probably a more realistic model of collision avoidance in this case. As well, soft constraints are not always satisfied. If we let $p(t)$ represent the trajectory of the foot, then the sensor trajectory, $s(t) = p(t) - d(1 - 4(0.5 - t)^2)$. The collision penalty term is then calculated as

$$E_{fcoll} = \begin{cases} K_{fcoll}(s_y(t) - y(s_x(t), s_z(t)))^2 & \text{if } s_y(t) < y(s_x(t), s_z(t)) \\ 0 & \text{otherwise,} \end{cases} \tag{5.2}$$

where $\mathbf{s}(\mathrm{t})$ is the sensor position at time t, and $y(x, z)$ is the height of the terrain at position (x, z).

We use a quadtree, as described in Chapter 3, to efficiently determine the elevation of the terrain below sensor $\mathbf{s}$ at time t.

Leg Length Constraints

The comfort constraints applied to the leg are identical to those applied in section 4.2.2, except we now consider a better estimate of the hip or shoulder position. The foot, $\mathbf{f_i}$, now attaches to the joint, $\mathbf{j_i}$ from Figure 5.2, rather than the hinge in Figure 4.14. Our calculation of E_{fcom} is

$$E_{fcom}(f, j) = K_{lc}(\|\mathbf{p_f} - \mathbf{p_j}\| - l)^2, \tag{5.3}$$

where $\mathbf{p_f}$ is the position of the foot, $\mathbf{f}$, $\mathbf{p_j}$ is the position of the joint, $\mathbf{j}$, l is the rest length of the leg, and K_{lc} is a weighting constant.

Physics Constraints

A physics constraint is useful for adding inertial effects to the motion of the leg. It has been suggested [7] [52] that natural motion can be generated by minimizing the amount of actuator energy required on the joint to achieve the motion. Because we do not have a complete leg model however, we cannot determine the applied internal joint torques. We could use inverse kinematics to generate a complete leg model, but this would significantly slow down our optimization procedure. As an approximation to a complete leg model, we treat the foot as a point mass attached to a massless virtual leg. We then evaluate the leg

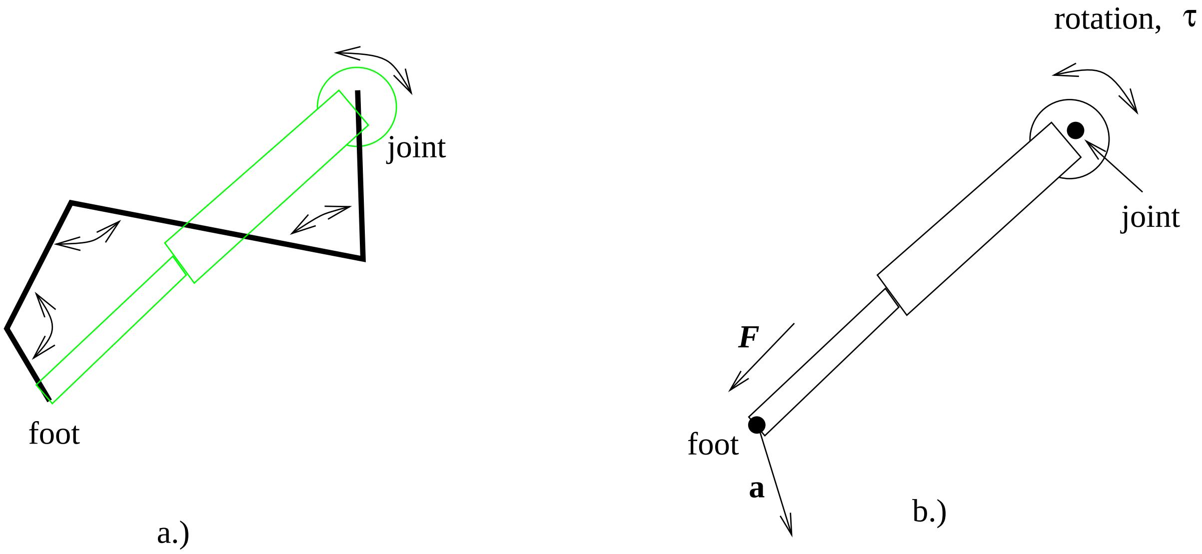


Figure 5.6: Virtual leg model used for physics constraints of the foot flight trajectory.

force and torque required to have the foot follow a given trajectory and introduce these into the optimization functional. An example of our approximate leg model is shown in Figure 5.6.

Assuming that the centripetal force is negligible (i.e., we have low angular velocities), the force required to achieve the necessary leg contraction or extension is given by

$$F_{compress} = m_{foot}(\mathbf{a} \cdot \mathbf{l}) \tag{5.4}$$

where $\mathbf{a}$ is the acceleration of the foot, $\mathbf{l} = \frac{(\mathbf{p}_{hip} - \mathbf{p}_{foot})}{\|\mathbf{p}_{hip} - \mathbf{p}_{foot}\|}$, and m_{foot} is the mass of the foot. We choose its contribution to the functional as:

$$E_{compress} = k_c F^2_{compress}, \tag{5.5}$$

where k_c is a constant used to scale the penalty. The torque of the virtual leg is calculated by $\tau = \|\mathbf{r} \times \mathbf{F}\|$, where $\mathbf{F}$ is the force applied on the foot, and $\mathbf{r}$= $\|\mathbf{p}_{hip} - \mathbf{p}_{foot}\|$. The torque's contribution to the functional is given by

$$E_{rotate} = k_\tau \tau^2, \tag{5.6}$$

where k_τ is a constant used to scale the penalty.

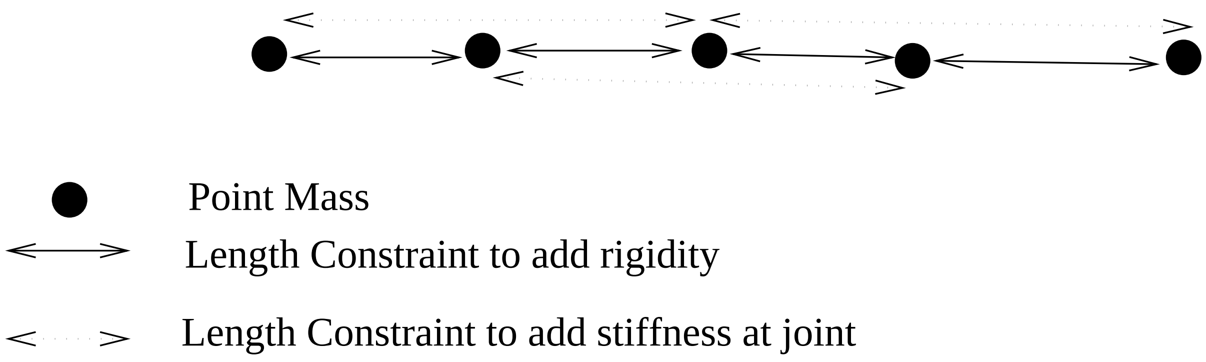


Figure 5.7: Point-mass model of a chain used for animating the tail and fur.

5.2 Secondary Dynamics

Secondary dynamics is the physically-based motion that occurs on normally stationary objects as a result of external forces applied on them. The motion is passive in the sense that the objects themselves do not actively invoke forces, they merely respond to external forces applied to them. The motion is secondary because we treat the reactionary forces that the object applies as though they were negligible. These motions are difficult to recreate kinematically. Passive dynamics adds visual cues to the motion that makes it easier to detect the momentum of the articulated figure. Good areas for applying dynamics include the tail, the fur, and the ears of the animal.

Our system applies passive secondary dynamics to animate the motion of the tail and the hair tufts of our animal. For the tail itself there are two levels of secondary dynamics: the body's motion drives the dynamics of the tail, while the motion of the tail drives the motion of the fur on the tail.

The tail and each strand of hair is modeled using a chain of point masses, as shown in Figure 5.7. Soft constraints are used to maintain the relative distances between adjacent point masses. A secondary set of soft constraints adds stiffness to the entire structure. We desire the tail and hair tufts to bend but maintain their lengths. An approximate dynamics algorithm, similar to that described by Overveld [62], calculates the motion of the point masses. A description of this approximate dynamics algorithm is included in Appendix C. While a spring mass system could also have been used, our approximate dynamics system exhibits less oscillation and we have found it to be more stable in

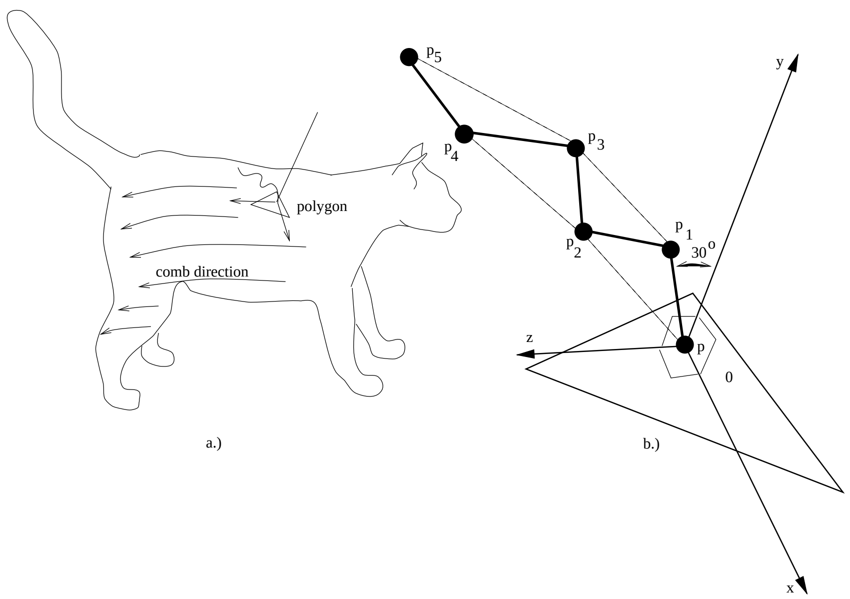


Figure 5.8: Hair model. (a) shows a polygon with a hair growing out of it placed on the body of the animal. (b) shows a closeup of the hair, the local coordinate system, and the length constraints.

practice for our application. Another possible approach for animating the hair and tail is to use spline curves rather than long chains of point masses [49].

5.2.1 Dynamics of the Tail

One end of the tail is fixed to the pelvis of the animal. The length constraints are set to make the tail straighten in the absence of external forces. As the body moves, the tail bounces and bends at the joints. Collision constraints are enforced for each point mass so that the tail does not penetrate the ground.

5.2.2 Dynamics of the Fur

There are many proposed models for animating hair. The method we use to animate the hair is similar to that described in [42] and [53]. After wrapping a polygonal mesh around the body of the animal, we attach one hair tuft to every polygon on the animals surface. The local coordinate system is established for each tuft as shown in Figure 5.8. The first point mass is constrained to remain in the center of the polygon. A constraint is used on the second point mass to comb the hair to the back of the animal by 30^o. Shortening the distance of the length constraints enforcing stiffness achieves a wavy appearance in each tuft of hair. Collision detection is used to keep the hair above the infinite plane associated with the polygon. We also use collision detection on the point masses of each hair to keep the hair from penetrating the ground.

5.3 Inverse Kinematics

When the positions of both ends of a limb or body part are known, inverse kinematics (IK) can be used to define the angles of the intermediate internal joints. Opportunities to make effective use of IK include the limbs and spine of the animal.

There are many possible ways to solve IK problems, the most well known of which involves using the pseudo-inverse of the Jacobian as described by Girard [18]. This method is effective, although it can be slow for parts with many joints due to the frequent recomputation of the pseudo-Jacobian matrix. The method effectively implements a least-squares joint movement. An improvement would provide more control over the IK solutions such that the motion of the joints closely matches that of the animal to be animated. It has been noted by others [31] that correct motion of leg joints during the kinematic reconstruction has a significant inpact on the visual effect of the animation.

5.3.1 Example-Based Inverse Kinematics

One possible solution to deal with the difficulties of controlling the IK solution is to use what we shall call *example-based inverse kinematics*. Example-based inverse kinematics is an IK system that uses user-defined keyframe examples to define the types of desired solutions. Given an IK query, interpolation is performed between the closest appropriate keyframes to produce a solution. A key feature in our approach is that the IK solutions produced are not only a function of the leg lengths but also the leg angle with respect to the body. This allows additional, region-dependent IK solutions, for example, a leg which maintains a constant length while swinging forward can potentially use different IK solutions during the swing.

Our example-based kinematics system is built on a 2D cylindrical coordinate system, as shown in Figure 5.9. The origin is located at the joint socket (hip or shoulder joint), and the two vectors which define the plane, $\mathbf{P}$, are foot direction, $\mathbf{d}$ and the vector

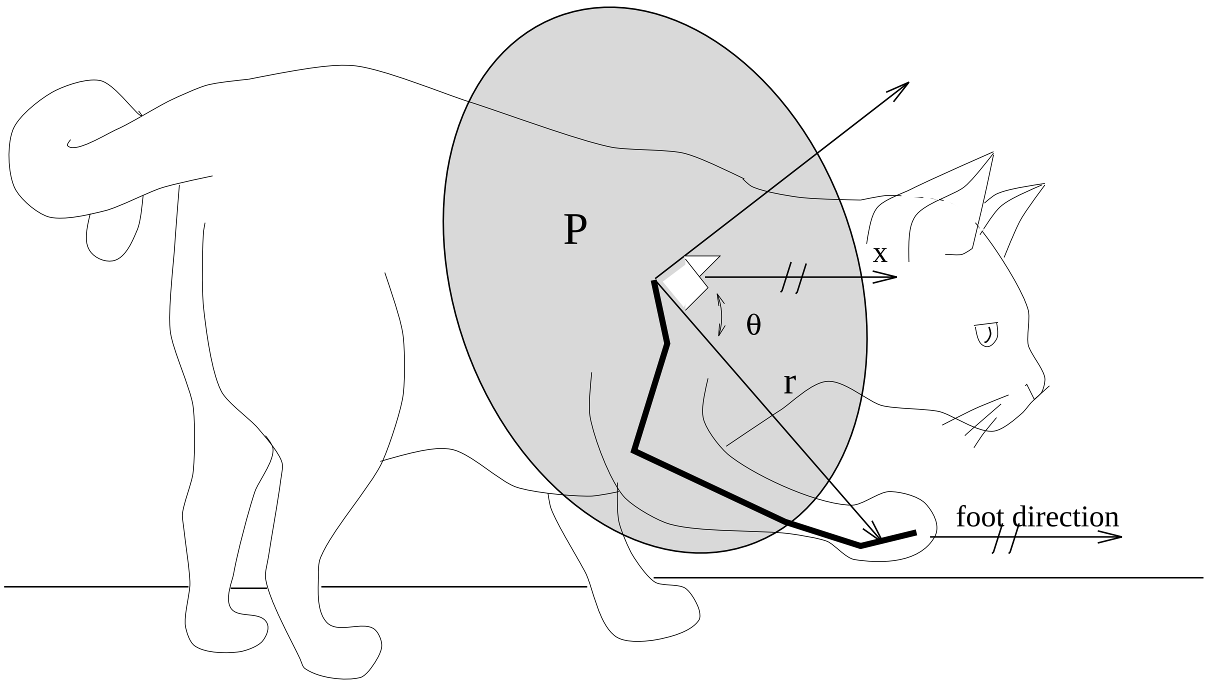


Figure 5.9: Coordinate system for plane that leg skeleton is placed in.

connecting the socket to the toes, **r**. The coordinate system is set such that the x axis points in the same direction as the foot, and the z axis points outwards. The IK is accessed through a procedure that takes in a query, (r, θ), and returns a keyframe, k, whose end effector position is r units away from the origin, and whose end effector angle on the plane equals θ.

The keyframe, k, is calculated by iteratively interpolating among several user-defined keyframes whose end effector positions surround the end effector position of the desired keyframe.

5.3.2 Definitions

Before we begin describing our procedure, we define the terminology and data structures that will be used to describe the method.

A *keyframe* is a data structure that stores the internal angles for a particular leg position. In Figure 5.10, we show an example keyframe for the hind leg of a quadruped. We store all of the internal angles, α_i, the angle of the end effector, θ, and the length of the end effector, r.

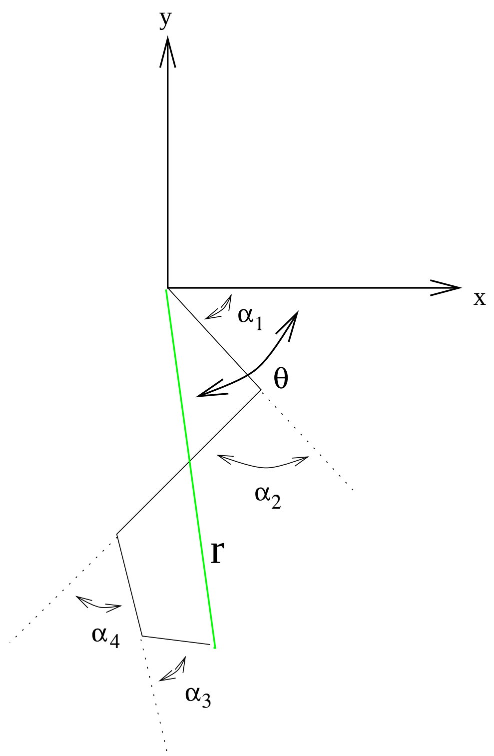


Figure 5.10: The keyframe data structure.

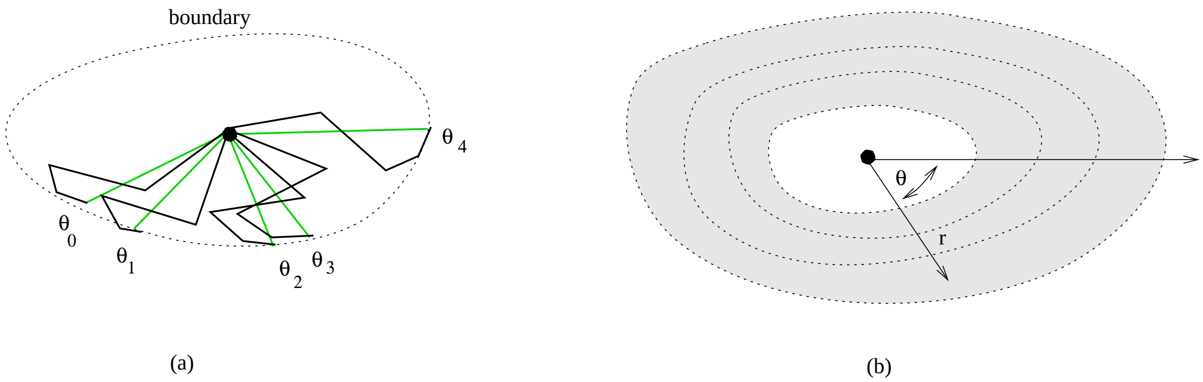


Figure 5.11: (a) A sequence of keyframes. (b) An *IK region*.

A *sequence of keyframes* is an array of keyframes ordered by θ, as shown in Figure 5.11(a). The members of the set can have a varying leg length, r. By interpolating joint angles of adjacent keyframes, a continuous series of IK solutions can be defined.

In Figure 5.11(b) we show the boundaries of several sequences of keyframes. If the sets themselves can be ordered with respect to the leg lengths, r, then we can interpolate between the keyframes on the boundaries to generate a keyframe for every pair of (θ, r) values within the innermost and outermost boundaries. We call this our *IK region*.

The steps that we use to calculate an element within this region can be summarized formally as:

- *input*: θ, r
- *output*: Keyframe, k, whose end effector position is θ and whose end effector length is approximately r.
- *algorithm*:
  1. For each set of keyframes, K_i, interpolate in θ to get k_i
  2. Iteratively interpolate between all k_i's to get a keyframe, k such that k's virtual leg length equals r.
  3. Adjust α_1 in k so that k's end effector position equals θ.

In our system, we use three sets of keyframes as shown in Figure 5.12. The set on the left, K_1, defines keyframes in a compressed position, the middle set, K_2, defines keyframes at a normal distance from the body, while the final set, K_3, defines keyframes of extended legs.

Given an angle, θ, we interpolate within a set of keyframes K_i by finding two keyframes, k_{i1} and k_{i2}, in the set whose end effector angles, θ_1 and θ_2, bound θ. The parameters of the interpolated keyframe, k_i, is linearly interpolated as follows:

$$k_i = (1 - \frac{\theta - \theta_1}{\theta_2 - \theta_1})k_{i1} + \frac{\theta - \theta_1}{\theta_2 - \theta_1}k_{i2} \tag{5.7}$$

Now, we have three interpolated keyframes, one from each set. We do another interpolation between the three sets to get a keyframe, k, whose end effector length is equal to r.

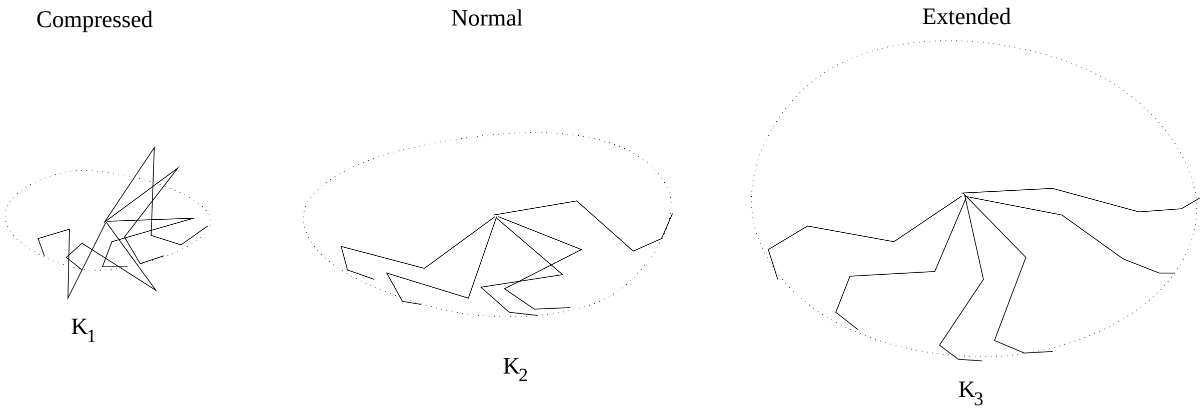


Figure 5.12: The three sets of keyframes required for our example-based inverse kinematics system.

We do not know what combination of the keyframes would give us the desired keyframe, so we do the interpolation iteratively until we find a keyframe one whose length matches r. In our implementation, we use a quadratic interpolation through the three keyframes. A linear interpolation would also work, although there might be a discontinuity in the rate of change of the internal angles as the leg crosses the middle boundary. Linear interpolation would be desirable if there is significant ringing in the quadratic interpolation. However, we have not noticed any ringing in our experiments. A quadratic interpolating polynomial, $s(t)$, is defined such that

$$s(0) = k_1, \tag{5.8}$$

$$s(0.5) = k_2. \tag{5.9}$$

and

$$s(1) = k_3. \tag{5.10}$$

The function $s(t)$ is derived using a Lagrange interpolating quadratic [28] to get

$$s(t) = k1\frac{(t-0.5)(t-1)}{(0-0.5)(0-1)} + k2\frac{(t-0)(t-0.5)}{(1-0)(1-0.5)} + k3\frac{(t-0)(t-1)}{(0.5-0)(0.5-1)}. \tag{5.11}$$

We iteratively calculate the spline interpolation using a binary search on t until we discover an interpolation whose distance from the body equals r. Currently, we are looking at solving for t directly. In any case, the values we use for the internal joint angles of the leg come from the resulting keyframe. Finally, we adjust angle α_1 of the resulting keyframe, k, so that the end effector angle is exactly θ. If the current end effector angle is given by θ', we adjust the final angle by adding $\theta - \theta'$ to α_1.

5.4 Procedural Kinematics

Procedural kinematics techniques are best applied to body parts whose motion is cyclical, well understood by the animator, or not clearly noticeable. We classify all methods not described by sections 5.1 to 5.3 in this category. This form of animation is often the fastest and easiest to implement, although it is difficult to achieve realistic motions if used too liberally. This category describes many possible techniques, of which we describe only a small subset.

An example of procedural animation is to use sinusoidal functions to modulate the motions of certain body parts, such as the shoulders and pelvis. Sinusoids can also be used for the head and tail motion and combined with randomization, as in [44], to add personality to the animation. If more information is available, such as the position of other body parts or the future motion of the animal then this can also be used to further constrain the positions of the remaining body parts. We apply such an approach for the animation of the head, pelvis, and torso of the animal.

5.4.1 Head

The head is constrained to turn of the direction of motion. We use the trajectory of the front point mass in the body of the animal to determine where the front of the animal will be at the designated timestep in the future, as shown in Figure 5.13. The head is turned in that direction, using a damped motion.

5.4.2 Torso and Pelvis

The pelvis and torso are animated once the motion of the feet is completely specified. The relative positions of the feet, with respect to the body are used to make the spine arch appropriately upwards. A more detailed description of the kinematic procedures to bend the torso and pelvis of the animal is contained in Appendix D.

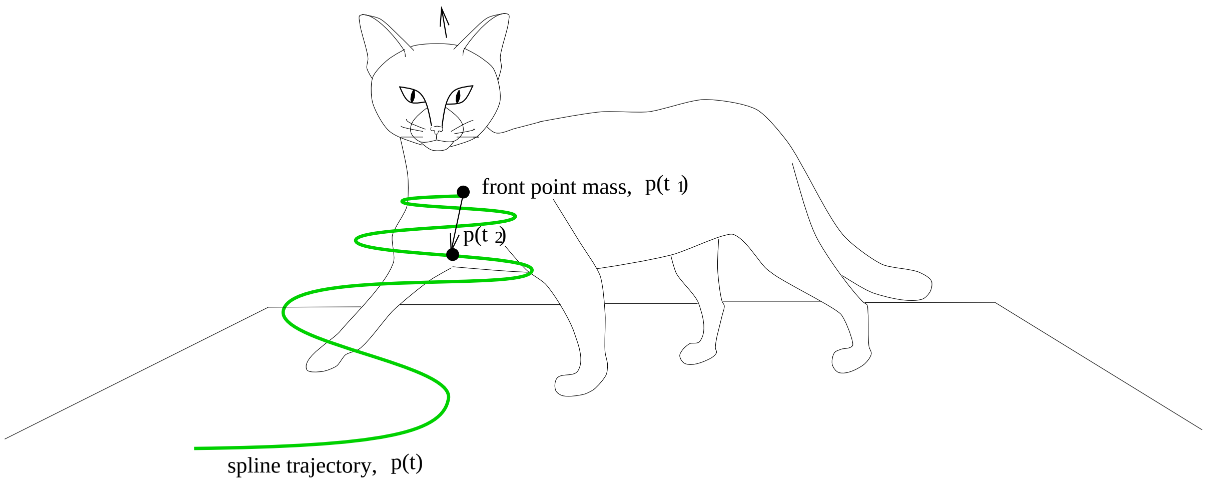


Figure 5.13: Calculating the direction of the head

Chapter 6

Results

This chapter describes various example animations produced using our animation system. A variety of gaits are examined in order to test the robustness of the system. Some of the tests include variable terrain locomotion. An example of autonomous motion is given. Finally, we test the generality of the method by applying it to a different character (a biped), with a different footprint type (a skating track). Lastly, we summarize what we have learned about the use of footprints as an animation primitive and make comparisons between our animation method and other techniques.

6.1 Hoop Walk

Our first test requires the animal to step through two rows of tires while walking. Although the underlying gait used is the same as for a standard quadruped walk, the actual motion is more complicated because extra precision is required to place the footprints. The objective of this test is to show how a complicated walk can be generated, the subjective quality of the motion, and how the collision avoidance and collision simulation impact the feet and the hair respectively.

The footprints for this motion are specified as shown in Figure 6.1. The timing parameters for each foot follows from the fixed pattern shown in Figure 6.2. No manipulation of the footprint positions was required to get the final animation to move correctly, other than the placement of the initial footprints.

After optimization, the resulting trajectories for the body point masses are as shown in Figure 6.3. Note that even though the motion specification is simple, the trajectory

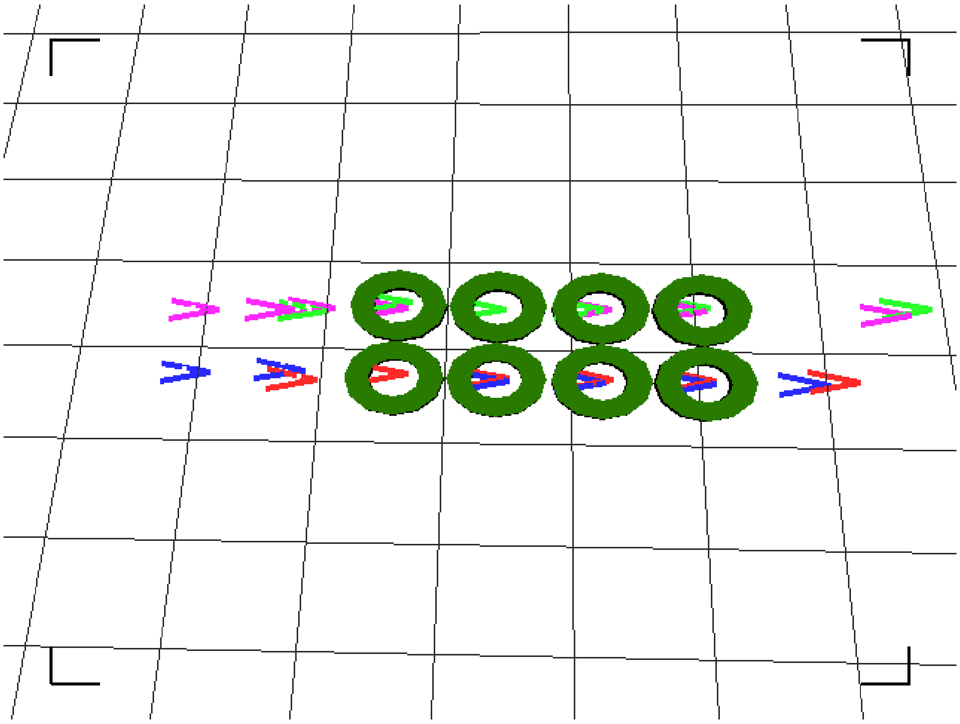


Figure 6.1: Footprints used in “Hoop Walk” animation.

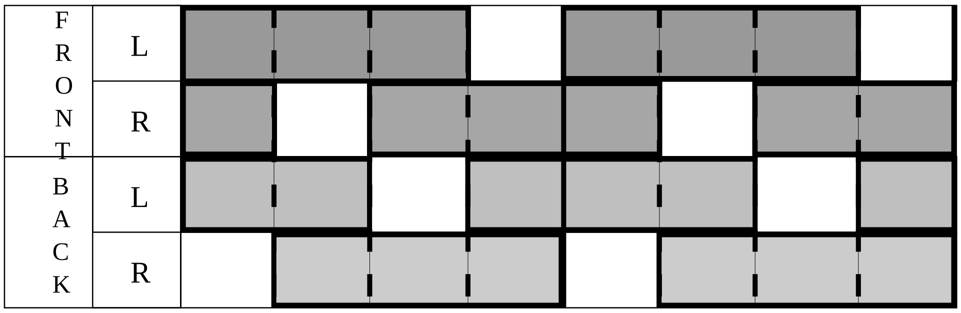


Figure 6.2: Footprint gait used in “Hoop Walk” animation.

optimization produces motions with vertical oscillations, which add character and realism to the motion (subjectively speaking). These motions occur as a result of both the shifting of weight to different legs and the short falling phases that occur on each half of the body during stepping. Such motion characteristics, which emerge naturally from our synthesis technique, would likely require considerable effort to generate using traditional animation techniques.

In Figure 6.4 we see some frames from the final animation. As each foot steps into or out of a tire, it carefully moves over the tire to avoid a collision. After every step into

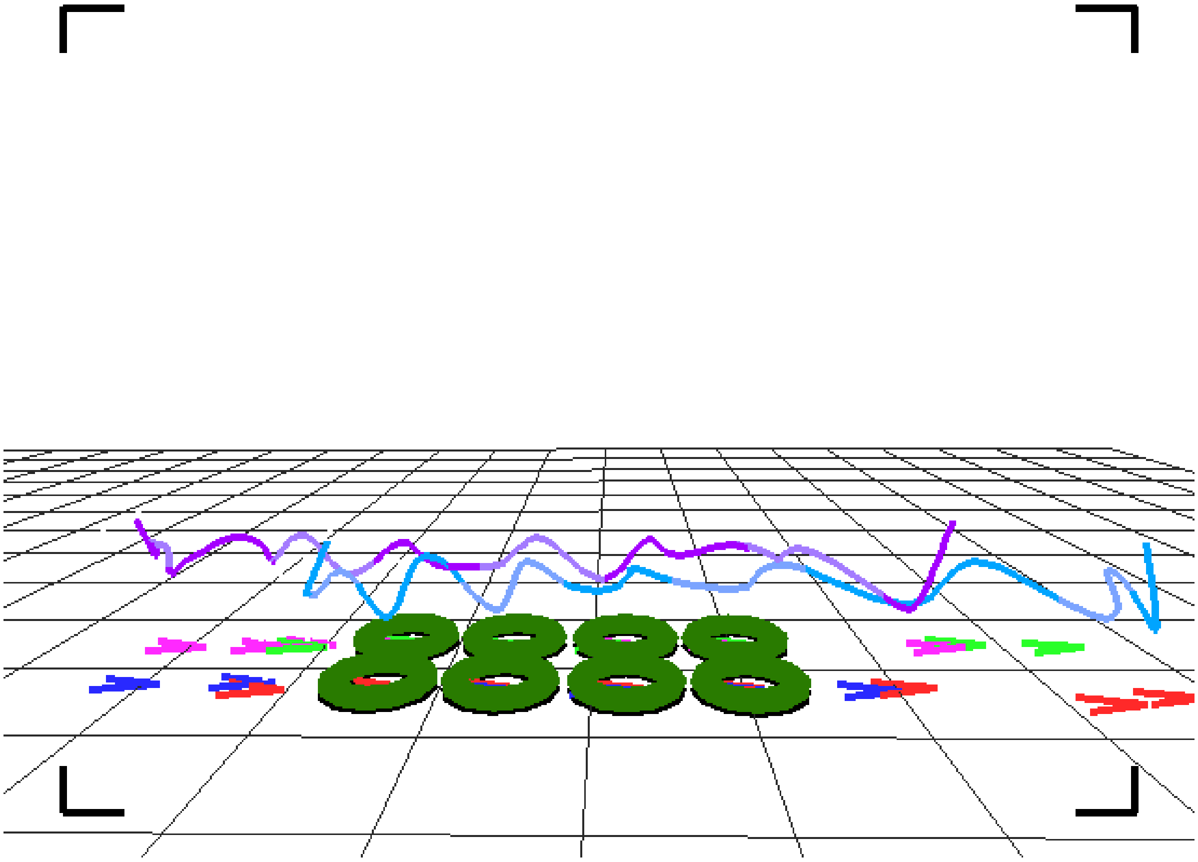


Figure 6.3: Optimized trajectories from "Hoop Walk" footprints.

a tire, the tire is obscured by sliding, moving hair which still has momentum from the stepping motion. Collision constraints on the hairs prevent them from penetrating the tires or the ground. This causes hairs to pile on the floor or drape over the surface of the tires.

6.2 Gallop

Gallops are particularly difficult to automatically generate because they are very ballistic in nature. We do not know of any physics-based animation system which is capable of generating gallops. In this section we test our system on a set of footprints that specify a gallop. We will also examine how our system handles turns. The gait parameters that we use in our gallop example is shown in Figure 6.5(a). As can be seen in the figure, each state has two or fewer legs in contact with the ground and the gait includes a short flight

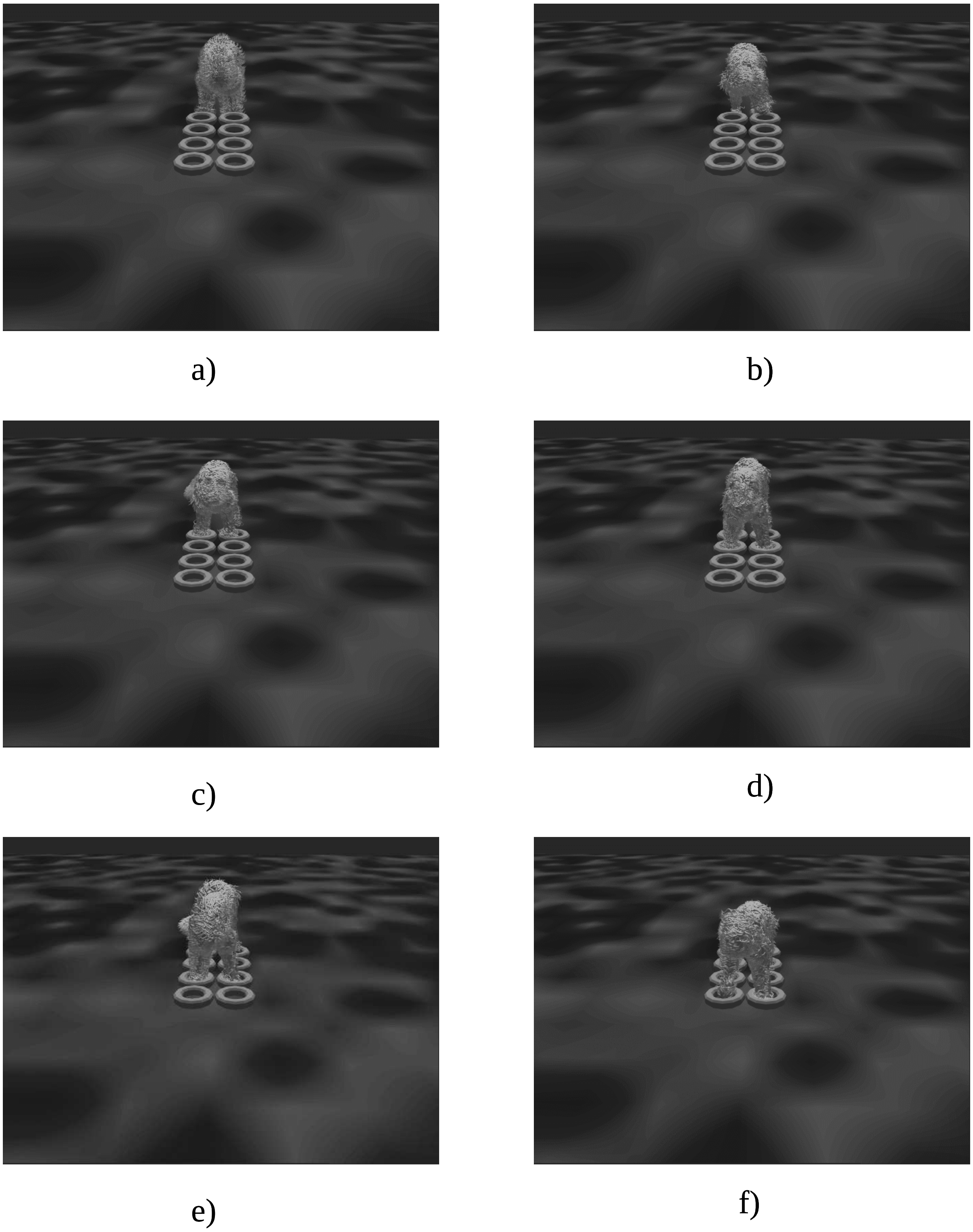


Figure 6.4: Frames in “Hoop Walk” animation.

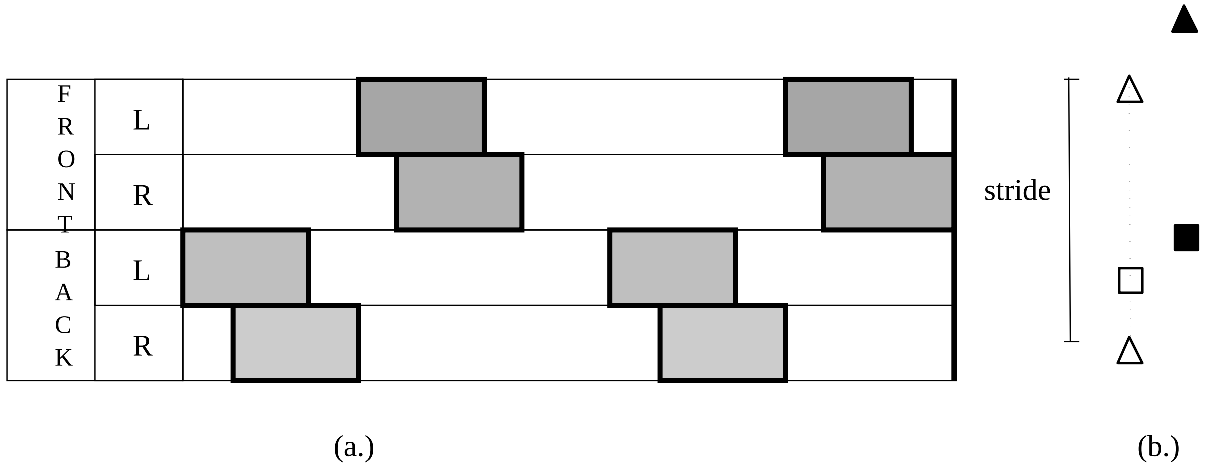


Figure 6.5: Footprint gait used in "Gallop" animation. (a) timing parameters. (b) suggested relative position parameters.

phase. Beside the gait parameters, we show the suggested relative footprint positions in Figure 6.5(b).

The footprints used in our gallop example are shown in Figure 6.6. Notice that there is a sharp turn in the middle of the set of footprints, which we have included to show how our system would handle turns at high velocities. In total, 48 footprints were used. All of the footprints were positioned manually with the help of the suggested pattern for the footprint positions. The stride length was increased during the footprint placement to simulate an acceleration in the gallop velocity. The timing information came from the gait parameters in Figure 6.5(a) and was not altered in the final animation.

In Figure 6.7 we can see that the trajectory optimization results in a motion that is both bouncy and smooth. The initial trajectory optimization took 75 seconds on a P150 rated[1] PC. The total optimization time for the footprint flight phases was 60 seconds. Reconstruction of the quadruped skeleton without the skin and fur is performed in real-time. Once the motion of the skin and fur are included, the reconstruction time is almost 60 seconds per frame for 11800 tufts of fur consisting of five control points for each tuft.

While generating this animation, we observed that occasionally the point mass trajectories were too high above the footprints, leaving the legs unable to satisfy the footprint constraints. This can result from weak comfort constraints, short footprint times, and

[1]A Cyrix 686 processor rated equivalent to a 150Mhz Pentium chip.

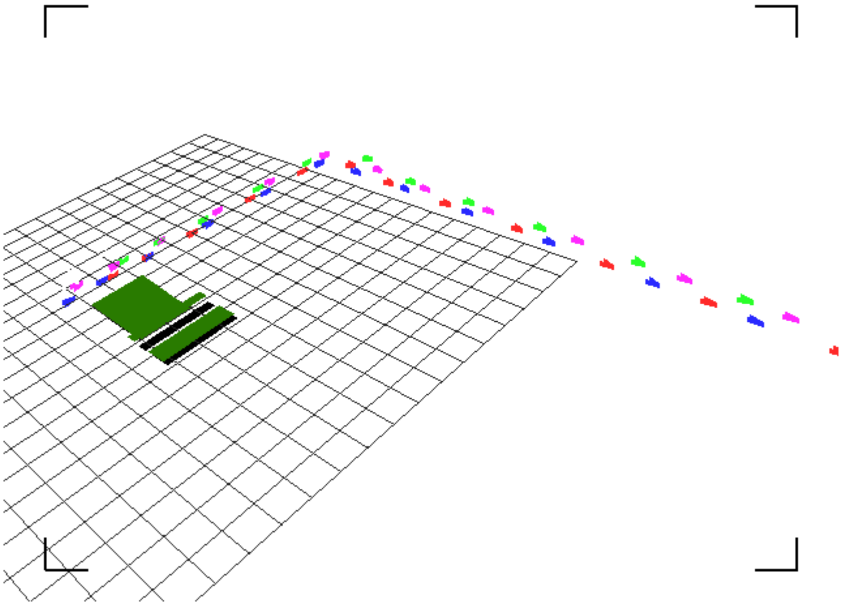


Figure 6.6: Footprints used in "Gallop" animation.

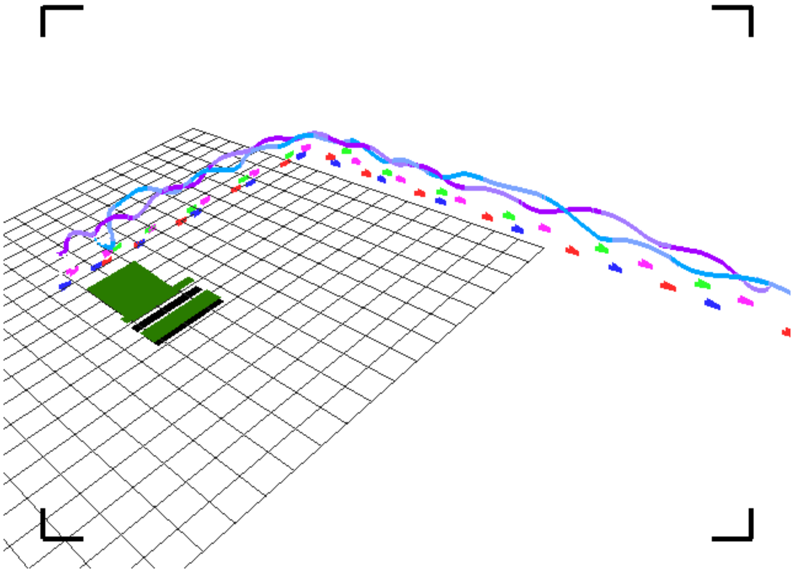


Figure 6.7: Optimized trajectories from "Gallop" footprints.

strong forces exerted by the spine if the two point masses get too close. Weak comfort constraints can be corrected by simply increasing the weight of the comfort constraint. Short footprint times relative to the entire gait cycle result in aliasing during our optimization calculation with coarse approximation timesteps, since our optimization functionals are calculated during discrete points in time. A solution for this is to increase the resolution where we sample the optimization functional. Finally, strong forces between point masses can occur if there are significant changes in the relative point mass distances. In our case, we placed the hind footprints too far ahead and the optimization caused the hind point mass to quickly move forward, which then pushed against the front point mass, lifting it upward.

We also noticed that the motion of the legs needed to be tuned. Before we tuned the animation, the legs moved slowly and had little inertia, which made the entire animation look stiff. To correct for this, we decreased the desired rest length of the legs during flight to 90% of the original value. This made the leg motion appear more elastic. We also decreased the strength of the physics constraints used on the legs to control the leg compression and torque. After implementing the changes in the leg motion, the final animation appears qualitatively realistic and no longer looks stiff.

Several frames of the final animation are shown in Figure 6.8. The legs appear to be push the body off the ground and then support it during landings. During the turn, the quadruped automatically tilts inwards in order to avoid toppling over, which is comparable to Girard's use of a special banking function in PODA [19].

6.3 Ramp Jump

The "Ramp Jump" animation tests the ability of our quadruped to maneuver over variable terrain. It demonstrates how our system responds to jumping motions and also highlights some of our system's collision detection.

Figure 6.9 shows the position of the footprints along with the trajectory optimization. The timing information for the jump is shown in Figure 6.10. The footprint data for the ramp jump was generated by directly positioning the footprints with a walking gait to set the timing information. The timing information was altered so that all of the feet

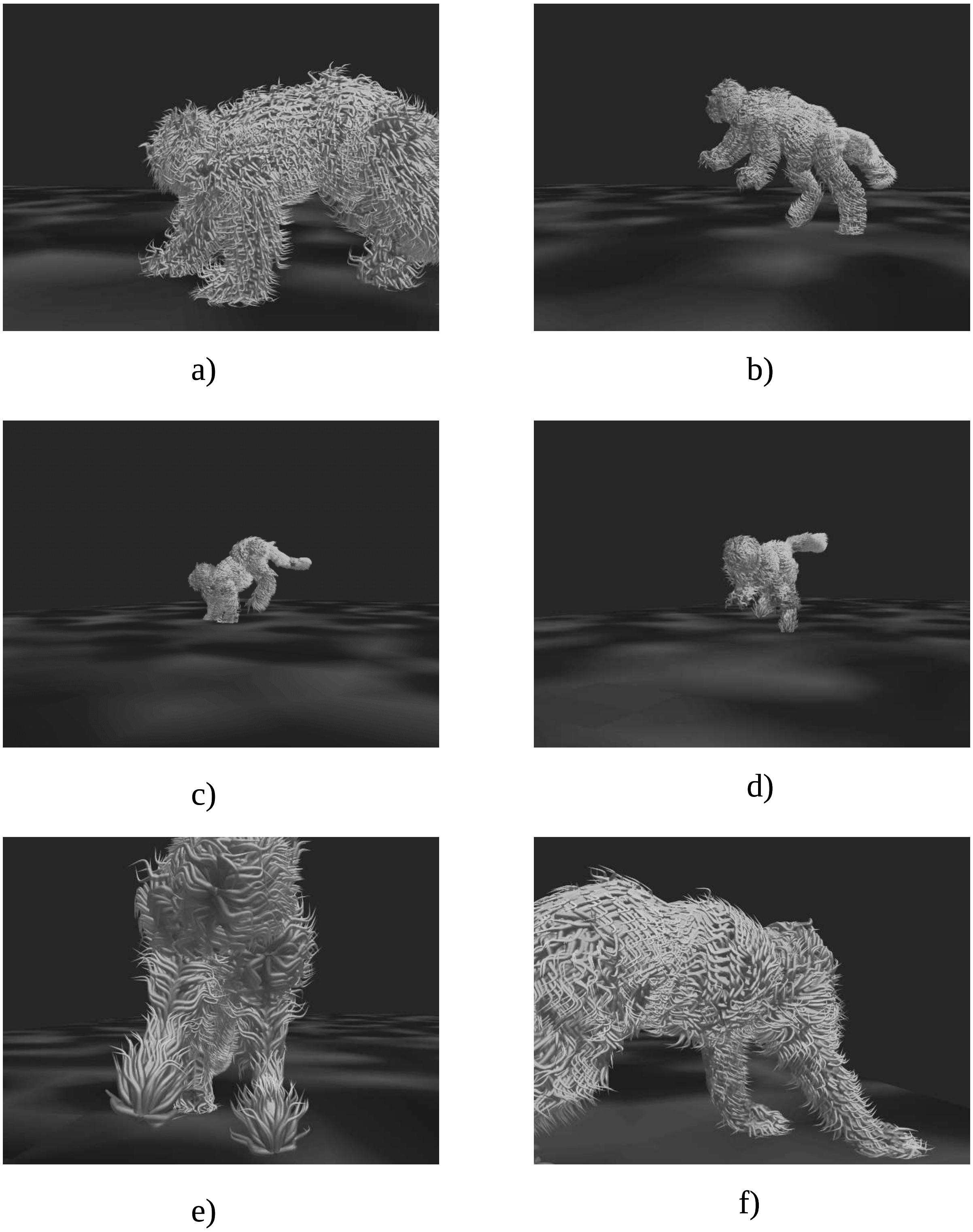


Figure 6.8: Frames in “Gallop” animation.

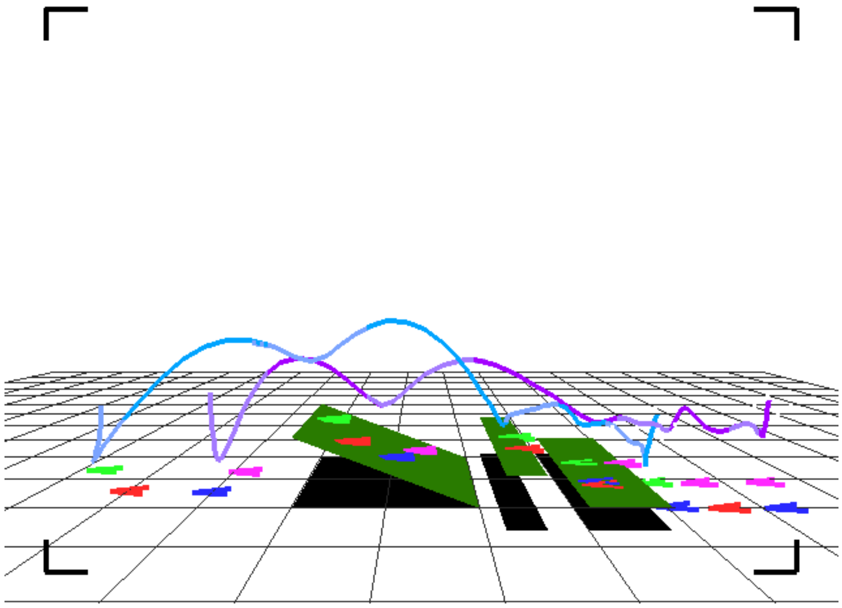


Figure 6.9: Footprints and optimized trajectories for “Ramp Jump” animation.

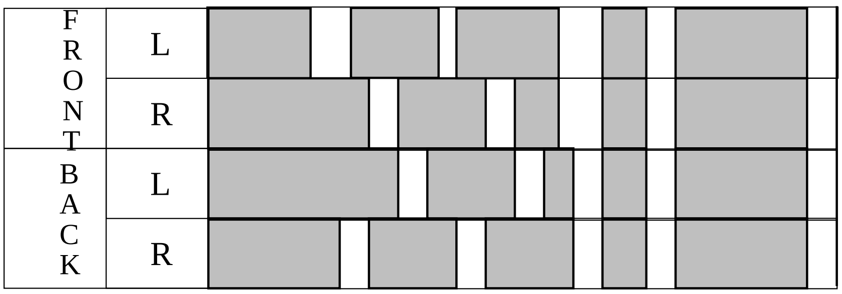


Figure 6.10: Timing information for footprints in “Ramp Jump” animation.

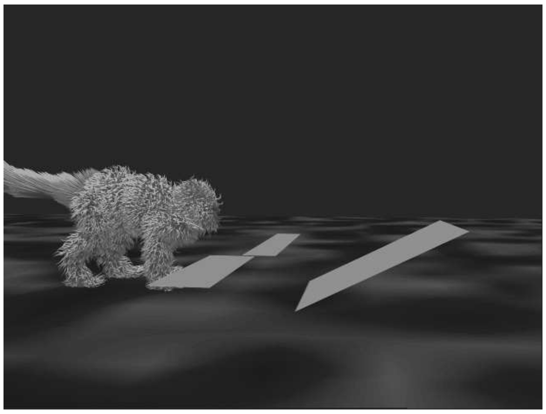
leave the ground simultaneously to generate the jump onto the ramp and similarly for the jump off the ramp. Notice that during the first jump, the hind feet leave the ground slightly after the front feet do. This was to avoid the possibility of the hind point mass surpassing the front point mass and awkwardly contorting the quadruped's body. In most situations this does not occur, but on some occasions we have had to adjust the footprint parameters to avoid this effect. An additional comfort constraint might also be used to penalize such positions, although we have not developed one.

The optimization step, which takes 60 seconds to compute on a P150 rated PC, generates a pair of arcs in each trajectory. The trajectory segments preceding and following the jumps are representative of the anticipation and follow through. The jump off the ramp is very high, and causes the body to touch the ground during the landing. Our point mass body constraints prevent the quadruped from penetrating too deeply into the ground. The collision constraints on the tail cause it to brush against, but not through, the surface of the ramp. As the quadruped walks up the stairs, its fur realistically hangs off the edges of the steps. When the tail brushes the surface of the ramp, the fur on the tail spreads out and brushes against the surface as well. In this animation there was no collision detection between the tail and the legs of the quadruped, although we have added this feature later, which we will demonstrate while discussing the "Pushups" animation in section 6.5. Several frames of our final animation are shown in Figure 6.11.

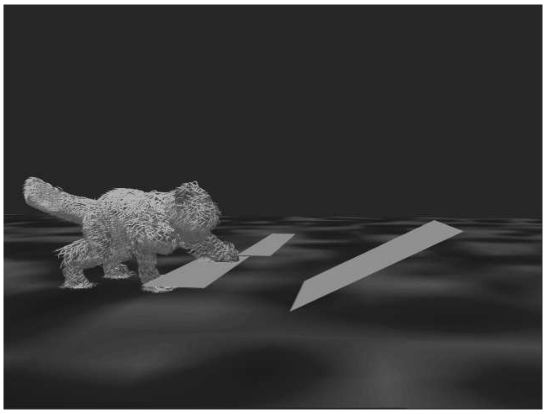
6.4 Autonomous Behaviour

Our next example tests the implementation of autonomous motion using our system. This will allow us to observe how our system performs over a long period of time. Variable terrain is used to show how our system responds to complicated environments. We will explain how the system was developed and describe the methods we have used to speed up the animation to interactive rates.

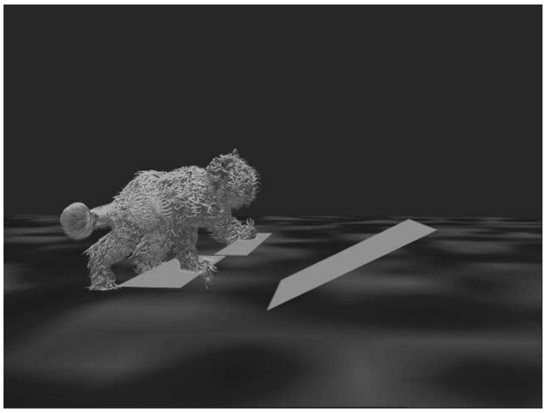
We have implemented a version of our animation system that generates autonomous motion at interactive rates. At each step we position the footprints using an automated process, as described in section 3.4, initialize the new spline control parameters, optimize the internal control parameters, and lastly display the quadruped motion defined by the



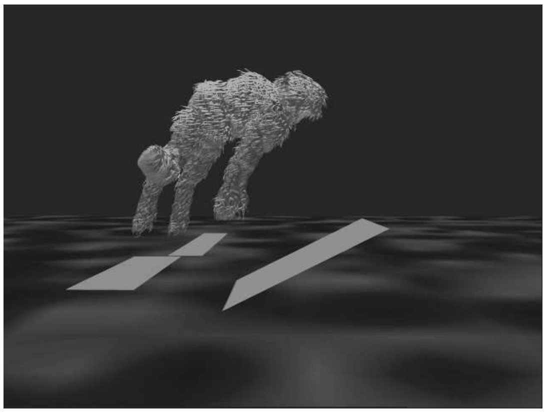
a)



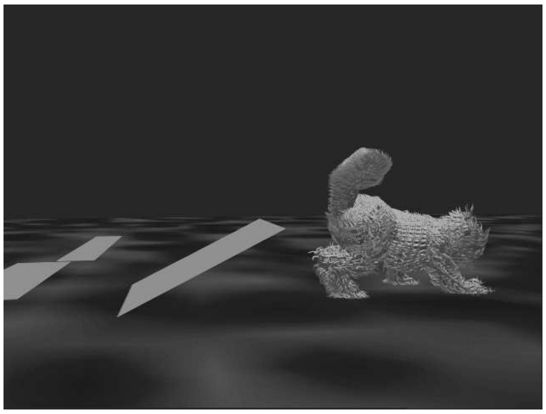
b)



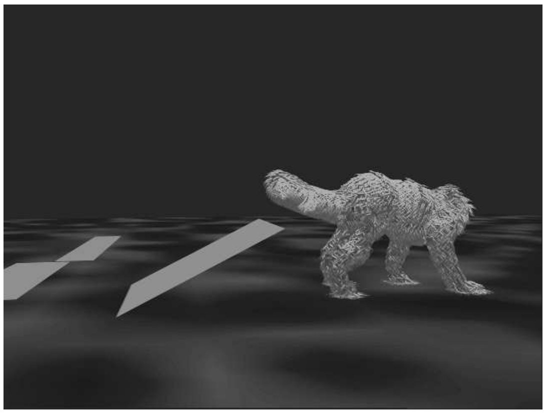
c)



d)



e)



f)

Figure 6.11: Frames in “Ramp Jump” animation.

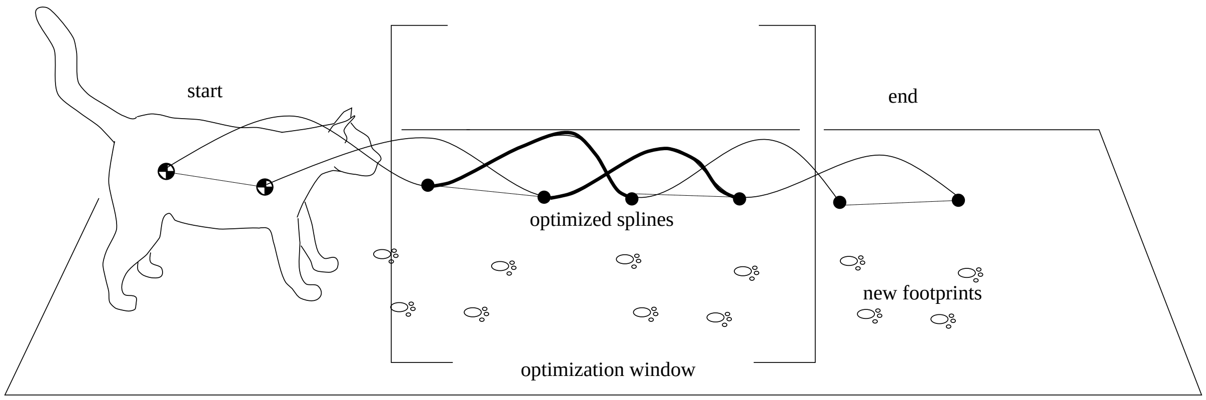


Figure 6.12: Autonomous motion example.

first spline segment. Because the skeletal reconstruction can be done quickly, we focus our efforts on improving the running speed of the optimization algorithm. One of the methods that we use is an optimization window, as shown in Figure 6.12. Our window is one spline segment wide. The optimization window speeds up the motion synthesis by restricting the scope of the optimization to particular segments of the trajectory. We have also utilized low cutoff parameters in our optimization procedures and a lower resolution estimation of our optimization functional. Together, it takes approximately one second on a to do this type of optimization on a 250Mhz SGI Indigo2. As stated earlier, the optimization occurs after the quadruped has been animated for one spline segment. This causes the animation to pause for approximately one second after the quadruped takes four steps. This pause could be eliminated by continuously interleaving the optimization calculations with the animation display.

In Figure 6.13 we show a creature walking autonomously on variable terrain. One problem that we noticed with the autonomous version of our quadruped system is that occasionally the trajectory dipped below the surface of the ground after a large number of footprints were placed. This problem was corrected by increasing the penalty for the body collision with the ground. Increasing the optimization window corrects this problem too, but adds some overhead to the optimization time. Additional constraints on the angles that the body makes with the legs might also keep the body from toppling over. We suspect that the cause of this problem might be the values we use to initialize the last spline control parameter. Since we set the velocity and acceleration of that parameter

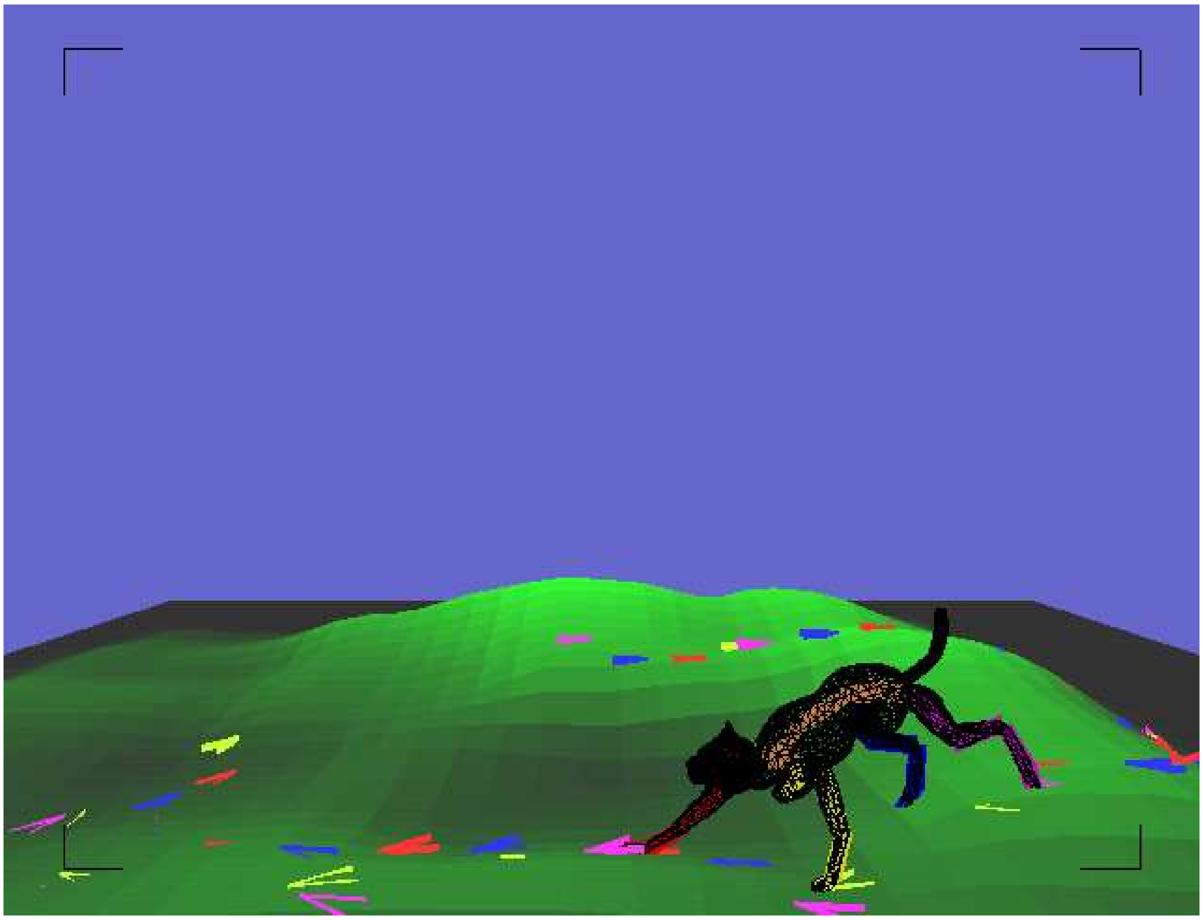


Figure 6.13: Autonomous motion example.

to zero, the optimization anticipates a stop at the end of the trajectory which dips the quadruped's body down and tilts the front upwards to slow the quadruped down. In most cases this does not affect the resulting animation because we animate only the first spline segment in the trajectory which is furthest away from the last control parameter and so is affected the least.

6.5 Pushups

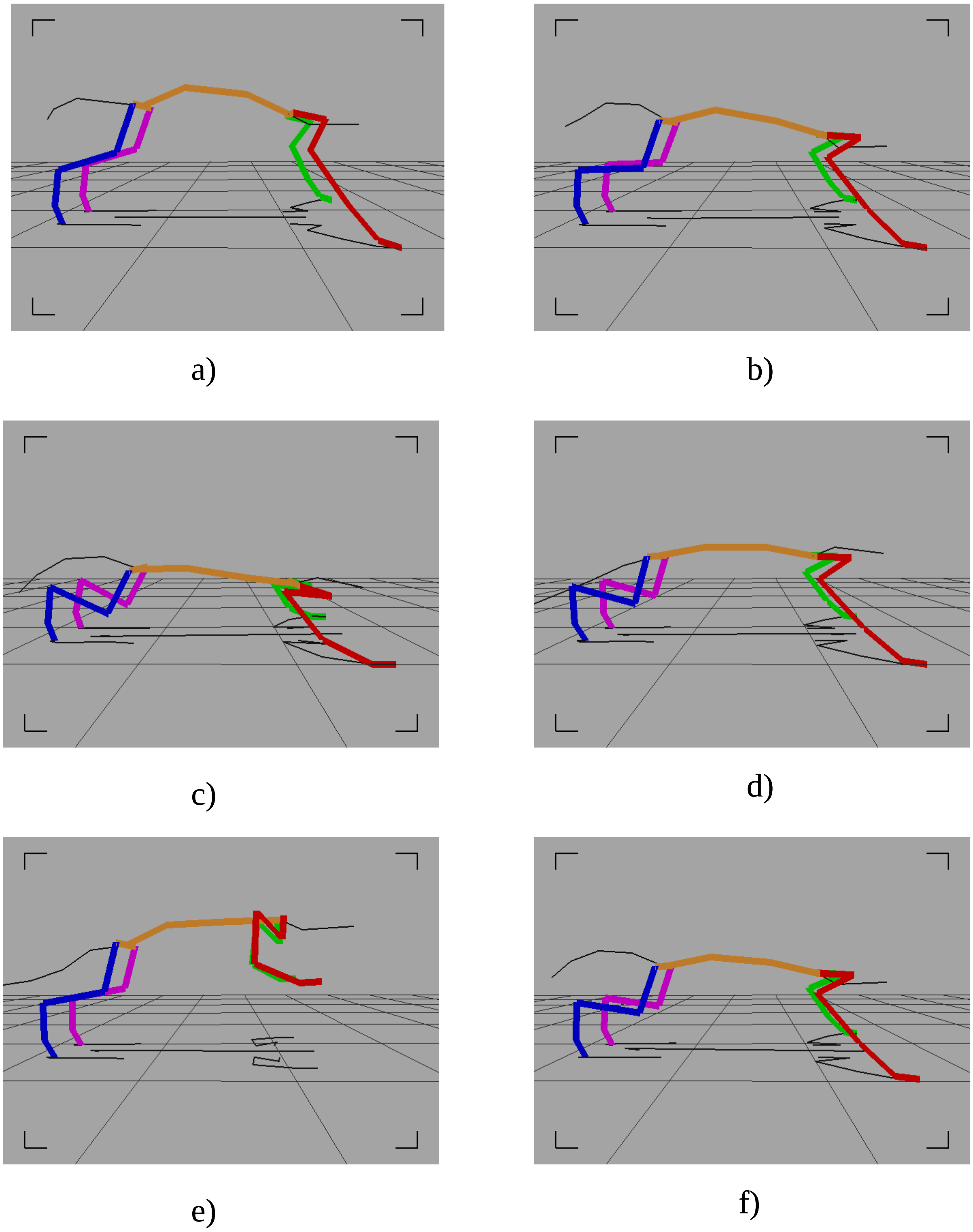
This example illustrates the use of footprints for motions which are not typical gaits, unlike walking or running, which are. In this example we can also see the effects of the collision detection that has been added on the tail. Again, the input is still footprints and timing. To generate the input, the hind feet are anchored to the ground for the duration of the animation, and the front feet are repeatedly raised off the ground and lowered.

In Figure 6.14 we show one cycle of the pushups. In the full animation the quadruped does five pushups, with each one looking realistic, but slightly different from each other because the initial conditions vary for each cycle. The tail does not penetrate the legs during this animation as happened during the "Ramp Jump" animation because of our additional collision constraint on the tail dynamics. The constraint keeps the tail from penetrating the plane defined by the point where the tail joins the body, and the heels of the hind legs, as is evident in Figure 6.14(d).

6.6 Skating Biped

In this example, we allow the footprints to move over time. Hermite curves are used to represent the sliding footprints of a skating biped. We chose to animate a biped to show that our techniques apply to a variety of creatures.

The motion of the biped is generated in exactly the same manner as for the quadruped, except we use only one point mass and procedurally animate the torso to bend in the direction of motion. To properly take the physics of pushing on a skate blade into account, one should also modify the physics to reflect that the foot reaction force should have no



a) b) c) d) e) f)

Figure 6.14: Frames in “Pushups” animation.

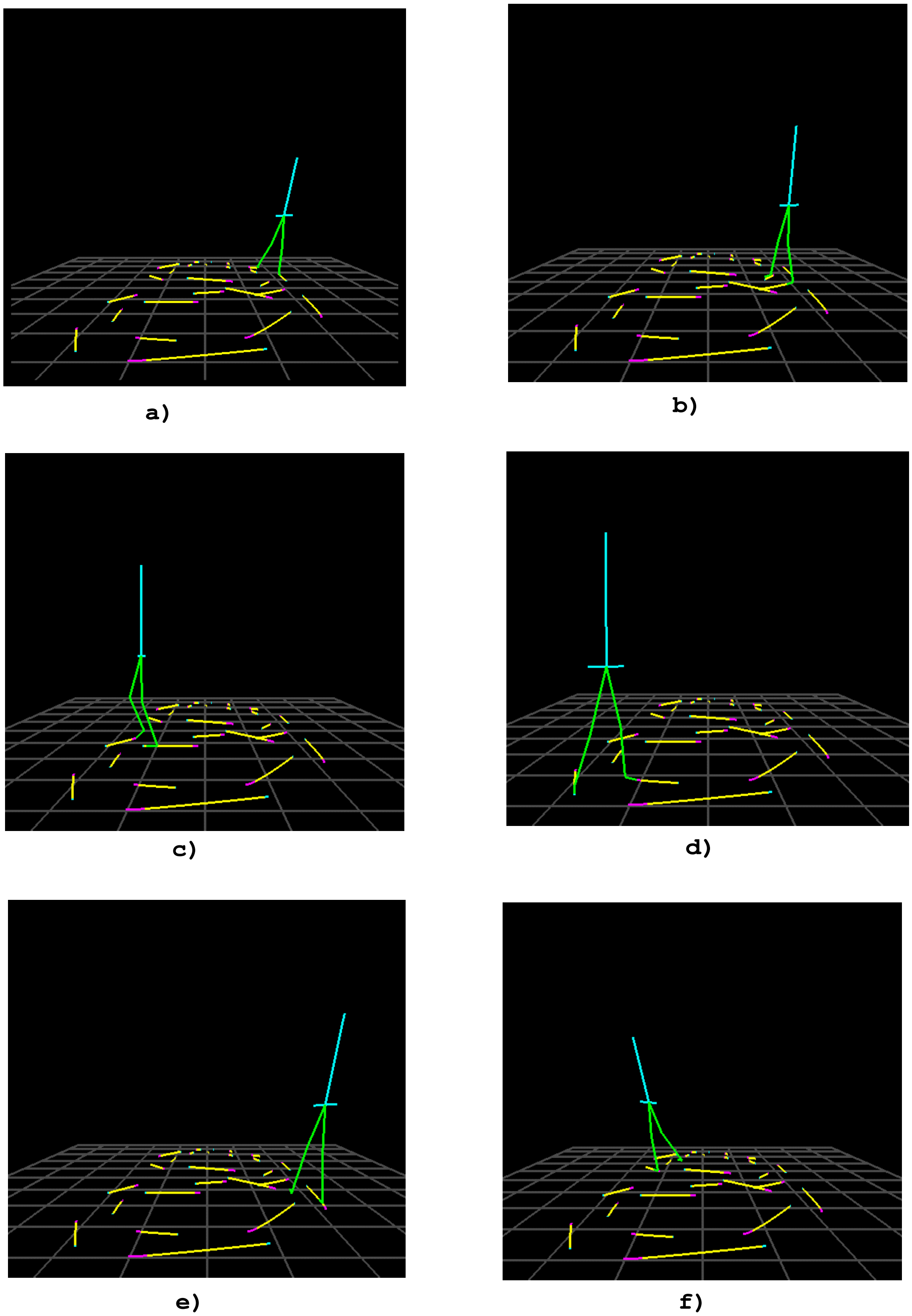


Figure 6.15: Frames in "Skating Biped" animation.

component which is tangential to the skate blade. We instead positioned the footprints almost perpendicular to each other and curved each skate track outwards to make it appear that the skate is actually pushing off the ground at the proper angle. Some of the frames generated from our skating example are shown in Figure 6.15.

6.7 Observations

Initially, we thought the most important aspect in footprint-based motion synthesis was the footprint positions. However, we have learned that the timing information of the footprints is very important in conveying realism. The motion is sensitive to the timing parameters. If the relative timing of the footprints is significantly different from that of the true animal, then the motion of the computer generated animal will appear awkward. Also, animals of different lengths walk at different rates. Smaller animals must raise and lower each foot quickly since their bodies are closer to the ground. However, in our experience, people do not have as strong an intuition of footprint timing as they do of footprint positions, and therefore a short learning curve is required in order to develop this intuition.

6.8 Comparisons

We conclude this chapter with a qualitative comparison between our method and other motion synthesis techniques that already exist for quadrupeds.

In comparison to the dog by Kokkevis et al. [31], our method appears to be capable of generating more complex motions. Like their method, ours is capable of generating walks and trots over variable terrain. However, their terrains are limited to slight inclines, whereas ours can be steeper and more variable. We have further shown that our method is capable of generating gallops and jumps. Finally, the motion produced using our method are considerably less stiff than those demonstrated in [31].

In comparison with Girard's work on PODA [19], our system implicitly accommodates all of the rules accounted for by PODA. The arcs of the trajectories in our "Ramp Jump" animation, and the bouncing trajectories in the "Hoop Walk" animation are clear examples of how our approach encompasses the effects accounted for by the vertical body motion procedures in PODA. Our system can also bank around corners without requiring a foothold adjustment, whereas PODA cannot. Both methods can generate good quality animations. However, it is not clear that PODA can readily accomodate motions such as skating, or that it is as flexible for adding additional hard or soft constraints to the motion. PODA can also not compromise on the physics during a flight phase.

The only similarities between our work and the work described in Girard's patent [20] are that both methods use footprints to specify the motion, and both use center of mass trajectories to represent the underlying motion of the figure. However, the patent mentions the use of keyframes which by definition restrict the pose of the articulated figure at various instances in time. We do not require any keyframes to be specified in our method. Also, our method is much more general than the work described in the patent in several ways. The footprints, for example, need not be static and are really a degenerate case of our skating tracks, where the splines specifying the motion of the foot are single points. Our underlying motion is specified by two point mass trajectories whose path is determined through optimization, whereas Girard uses one trajectory only, and does not use optimization techniques to calculate it. Therefore, all of the differences between our method and PODA also exist between our method and Girard's footprint-based method. Finally, our method works for characters with an arbitrary number of legs, while it has not been demonstrated that Girard's footprint-based method applies to anything other than bipeds.

Chapter 7

Conclusions and Future Work

We have presented a hybrid method which uses both physically-based and kinematic methods to animate articulated figures over a set of user-defined footprints. Our research has covered several areas and has given us some insights into how kinematic and dynamic techniques can be integrated. In this chapter, we summarize our contributions and future directions.

7.1 Contributions

7.1.1 Hybrid Techniques

Working with a complete model throughout the motion synthesis can greatly compound the difficulty of generating an animation because of the number of degrees of freedom to take into account. This complicates the motion synthesis as the system attempts to calculate the optimal motion for every joint. These methods do not reflect the likelyhood that real humans and animals do not have the time to compute optimal solutions, and in our opinion utilize a limited set of rules and experience in order to locomote.

We know that physics can be used to generate realistic motions, but we question its direct application to complex models. From our results it seems as though a complicated model is not always necessary. It is possible to generate much of the underlying motion of an articulated figure with a simplified model. This simplified model can remove redundancy that exists in a fully detailed model because of coordination among body parts. Our techniques exploit this to obtain the advantage of faster-converging, simpler optimization schemes for calculating the animal's trajectory.

The full skeletal model is restored afterwards by applying various kinematic, dynamic and optimization methods to the remaining joints and limbs. By not limiting ourselves to one particular technique for the entire motion synthesis, we are able to break up the motion synthesis problem into parts and apply various techniques on each part. This allows us to choose the most convenient animation method for each part, and to apply the various techniques available where they are best suited.

7.1.2 Optimization

The core of our animation system is the trajectory optimization of the simplified body. With it we achieve a balance between physical realism and kinematic constraints that simply is not possible with purely dynamic methods or kinematic methods. With optimization, the correctness of one constraint may be sacrificed to minimize the amount of deviations from the other constraint. This is an advantage because often the animator may require that the articulated figure perform actions that may be unrealistic in the real world. *Animation physics* is different than real physics [6] because physical principles are often exaggerated or violated in order to enhance the visual appeal of the animation. Therefore, any tool that can stretch the underlying physical principles in order to generate a realistic animation on exaggerated inputs would be a great asset for animators.

Two terms are calculated and summed together in our optimization functional: a physics penalty and a comfort penalty. Our physics penalty simply ensures that the optimal trajectory consists of accelerations that can be accounted for by our physical model. The resulting effect of the physics penalty is an animation where the articulated figure moves in the direction that the feet can push, and anticipates ballistic motions such as jumping by bending in the correct direction before the action. Because of our simplified body model, we are able to quickly determine reasonable regions of plausibility for the acceleration of the creature's body.

Our comfort penalty, on the other hand, ensures that the creature's motion does not contort into awkward positions. The actual definition of *comfort* is open, and can be defined in different ways. It is intended to capture some general motion characteristics.

The other motion characteristics are then captured with the reconstruction algorithms. In the implemented prototype, we have achieved good results with simple comfort constraints limiting the leg lengths of the articulated figure, and the closeness of the body from the ground. Other suggested comfort constraints include actual keyframes, and constraints on the angles of the virtual legs.

7.2 Footprints

We have shown that footprints are indeed a useful primitive for generating animations of quadrupeds. In doing so, we have demonstrated that it is possible to generate high quality animations of articulated figures by constraining a small subset of the available degrees of freedom. Complete user control of the articulated figure is not always necessary, and in fact may impair the users ability to generate quality animations by requiring an unreasonable amount of detail at the wrong point in the animation process.

7.3 Future Work

There are many possible future directions that we would like to explore. The most promising seems to be the application of trajectory optimization techniques on other types of movement such as powerlifting, skateboarding tricks, rock climbing, dancing and many more. All of these motions are very complicated, but there is some basic understanding about what drives the overall motion of the underlying structures. So it is possible to develop optimization functionals consisting of physics and comfort constraints for these motions which allow us to optimize the underlying trajectory of the motion, and reconstruct the final motion afterwards.

An interesting area to pursue is a method to analyze motion capture data for the purposes of generating simplified underlying structures, optimization functionals, and reconstruction methods automatically. This would allow us to greatly expand the number of animations that we can generate. If we can also generate a scheme that can generate continuous transitions between one trajectory optimization to a trajectory optimization of some other type of motion, then we would have a flexible and realistic animation system that responds correctly to changes in its environment.

The use of footprints as a user interface is another area potentially requiring further work. Our footprint generation system introduced the simplest method for footprint generation and manipulation. Besides improving the existing tools, we would like to be able to automatically generate timing information as a general function of footprint positions and an animal's skeletal structure. Research is required to develop a better optimization metric for automatic adjustment of footprint positions. A path planning algorithm would be useful as well. Imagine requiring for a set of footprints to make a given quadruped cross a stream without getting its feet wet, and having the computer automatically position the footprints on the stones in the stream.

In the future, we would like to develop a real-time implementation of our animation system. One possible approach is to generate the optimization concurrently with the animation. Our work with the autonomously walking quadruped has shown that we can calculate the optimization at interactive rates, so we simply need to integrate the optimization algorithm with the animation routine. We are also examining how to speed the optimization calculations by using a coarser optimization scheme, with the goal of developing a method that is fast enough to generate motions for characters in interactive games or virtual worlds. Some analysis on the statistics behind the optimization routines would also be useful. There may be some trends in the optimization that we may be able to exploit by allowing us to limit the search space of a global optimization. One method that might be useful would involve storing previously calculated examples on a database, and then using that data to guide the convergence of our optimization.

The reconstruction scheme is another area where more research is required. We would like to add a stochastic component to the motion, as in [44], to attempt to reproduce natural motion variation. With a stochastic model, we could add *personality* to the motion of the head so that it appears that the creature has some underlying intelligence. We could also simulate strain in the limbs by jittering them when they are in perceived unstable positions.

Appendix A

Derivation of Basis Matrix for Splines

In this section we derive the basis matrix for the splines used to represent the trajectories, as described in Chapter 4.

The free parameters used to represent the curve are the position, first derivative, and second derivative of the spline's end points. Let $s(t)$ represent the spline function, $t \in [0, K]$ where K > 0. The constraints used are:

$$s(0) = s_0 \tag{A.1}$$
$$\frac{ds}{dt}(0) = s'_0 \tag{A.2}$$
$$\frac{d^2s}{dt^2}(0) = s''_0 \tag{A.3}$$
$$s(K) = s_1 \tag{A.4}$$
$$\frac{ds}{dt}(K) = s'_1 \tag{A.5}$$
$$\frac{d^2s}{dt^2}(K) = s''_1, \tag{A.6}$$

where $s \equiv [s_0 \ s'_0 \ s''_0 \ s_1 \ s'_1 \ s''_1]$ represent the free parameters which define the curve.

There are six constraints in the spline definition, so $s(t)$ will be a degree 5 polynomial.

In matrix form, this can be defined as

$$s(t) = \begin{bmatrix} 1 & t & t^2 & t^3 & t^4 & t^5 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} \tag{A.7}$$

or

$$s(t) = T \cdot A. \tag{A.8}$$

The first and second derivatives of $s(t)$ can be expressed as

$$\frac{ds}{dt}(t) = \begin{bmatrix} 0 & 1 & 2t & 3t^2 & 4t^3 & 5t^4 \end{bmatrix} A \tag{A.9}$$

and

$$\frac{d^2s}{dt^2}(t) = \begin{bmatrix} 0 & 0 & 2 & 6t & 12t^2 & 20t^3 \end{bmatrix} A \tag{A.10}$$

respectively. We use equations A.9 and A.10 to rewrite the constraints of our spline as

$$G_s = BA, \tag{A.11}$$

where $G_s = \begin{bmatrix} s_0 \\ s'_0 \\ s''_0 \\ s_1 \\ s'_1 \\ s''_1 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 1 & K & K^2 & K^3 & K^4 & K^5 \\ 0 & 1 & 2K & 3K^2 & 4K^3 & 5K^4 \\ 0 & 0 & 2 & 6K & 12K^2 & 20K^3 \end{bmatrix}$. Then we solve for A in equation A.11 by using

$$A = B^{-1}G_s. \tag{A.12}$$

The spline equation thus can be expressed as $s(t) = T \cdot B^{-1} G_s$. In this form, G_s contains the control parameters and B^{-1} is the *basis matrix*:

$$B^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{-10}{K^3} & \frac{-6}{K^2} & \frac{-3}{2K} & \frac{10}{K^3} & \frac{-4}{K^2} & \frac{1}{2K} \\ \frac{15}{K^4} & \frac{8}{K^3} & \frac{3}{2K^2} & \frac{-15}{K^4} & \frac{7}{K^3} & \frac{-1}{K^2} \\ \frac{-6}{K^5} & \frac{-3}{K^4} & \frac{-1}{2K^3} & \frac{6}{K^5} & \frac{-3}{K^4} & \frac{1}{2K^3} \end{bmatrix} \tag{A.13}$$

We can also write

$$s(t) = [f_1(t)\ f_2(t)\ f_3(t)\ f_4(t)\ f_5(t)\ f_6(t)]G_s, \tag{A.14}$$

where $f_1 \ldots f_6$ are the basis functions of the spline curve. In our case, the basis functions are:

$$f_1(t) = 1 - \frac{10}{K^3}t^3 + \frac{15}{K^4}t^4 - \frac{6}{K^5}t^5 \tag{A.15}$$

$$f_2(t) = t - \frac{6}{K^2}t^3 + \frac{8}{K^3}t^4 - \frac{3}{K^4}t^5 \tag{A.16}$$

$$f_3(t) = \frac{1}{2}t^2 - \frac{3}{2K}t^3 + \frac{3}{2K^2}t^4 - \frac{1}{2K^3}t^5 \tag{A.17}$$

$$f_4(t) = \frac{10}{K^3}t^3 - \frac{15}{K^4}t^4 + \frac{6}{K^5}t^5 \tag{A.18}$$

$$f_5(t) = -\frac{4}{K^2}t^3 + \frac{7}{K^3}t^4 - \frac{3}{K^4}t^5 \tag{A.19}$$

$$f_6(t) = \frac{1}{2K}t^3 - \frac{1}{K^2}t^4 + \frac{1}{2K^3}t^5. \tag{A.20}$$

Finally, a piecewise continuous trajectory can be made from two such splines, $s(t)$ and $r(t)$ by making the end parameters of the first spline equal the starting parameters of the second spline, so that:

$$s_1 = r_0 \tag{A.21}$$

$$s_1' = r_0' \tag{A.22}$$

$$s_1'' = r_0''. \tag{A.23}$$

Appendix B

Initialization of Free Parameters

In this appendix we describe in detail the initialization procedure used to position the initial spline trajectories during our trajectory optimization procedure.

The spline segments are initialized by placing the center of mass, $\mathbf{x}$, over the average position of the feet and then positioning the two point masses, $\mathbf{p_1}$ and $\mathbf{p_2}$, in the same direction as the average foot direction, as shown in Figure B.1. At the instant in time that the nodes will be placed, we calculate the footprint position for each foot, as shown in Figure B.1a. If at that instant in time, t, the foot is not on the ground, then we approximate the footprint position using

$$pos = (1 - \frac{t - t_0}{t_1 - t_0})f_0 + \frac{t - t_0}{t_1 - t_0}f_1, \tag{B.1}$$

where f_0 is the last foot position before t, f_1 is the next foot position after t, t_0 is the takeoff time of the foot and t_1 is the landing time of the foot. Once an approximate position for each foot has been found we average the position to calculate the center of feet, which we label $\mathbf{c}$ in Figure B.1(b). We approximate the spine direction to be equal to the average foot direction, which is labelled $\mathbf{d}$ in Figure B.1(c). If h is the average of each leg's rest length, we set $\mathbf{x} = (pos_x, pos_y + h, pos_z)$. The relative weights of the two point masses are then used to position them along the line $\mathbf{d}$ that intersects $\mathbf{x}$, as shown in Figure B.1(d), where s is the rest distance between the two point masses. Therefore, the initial positions for $\mathbf{p_1}$ and $\mathbf{p_2}$ are

$$\mathbf{p_1} = \mathbf{x} - \frac{w_2 s \mathbf{d}}{w_1 + w_2} \tag{B.2}$$

$$\mathbf{p_2} = \mathbf{x} + \frac{w_1 s \mathbf{d}}{w_1 + w_2}. \tag{B.3}$$

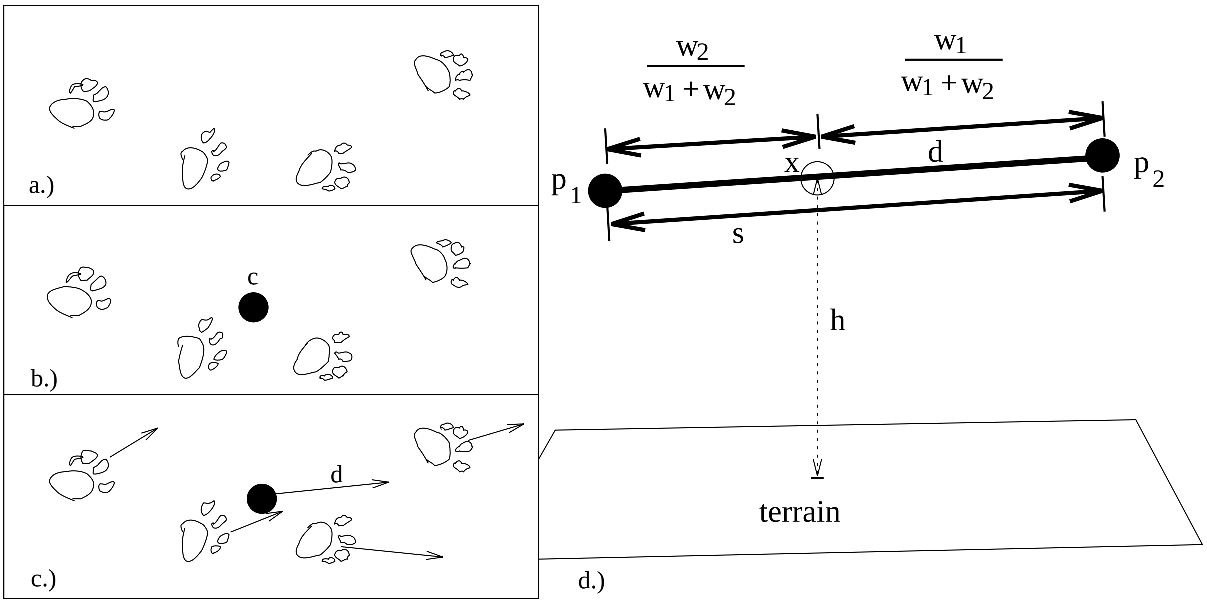


Figure B.1: Initialization of control point.

The initial accelerations and velocities at those node positions are set to 0.

Appendix C

Approximate Dynamics System

We use point masses to approximate the dynamics of the tail and fur. Our system differs from a spring mass system because we employ an iterative scheme which attempts to restore the relative distances between neighboring point masses after external forces cause the point masses to be displaced. This has the effect of maintaining the original lengths of the simulated chains without using the stiff spring mass system which would otherwise be required. As in spring mass systems, our approach can be applied to simple chain links of point masses, or more complicated models. A comparable method is discussed in [62].

We begin with a data structure that defines the object with point masses, and then initialize the constraints of each point mass. These constraints include *distance constraints* from other point masses, *position constraints* which fix points to a specific position in the real world, and *relative position constraints* which attempt to keep a point mass at a specific position relative to some local coordinate system.

Given a current configuration of point masses, three stages are required to calculate the next configuration in time: (1) a dynamics step where each point mass is animated under its own external forces, (2) a restoration step where the point masses are displaced in an attempt to restore the length constraints, and (3) a recalculation step where each point mass' new velocity is calculated. We show an example of these stages in Figures C.1(b-d) respectively.

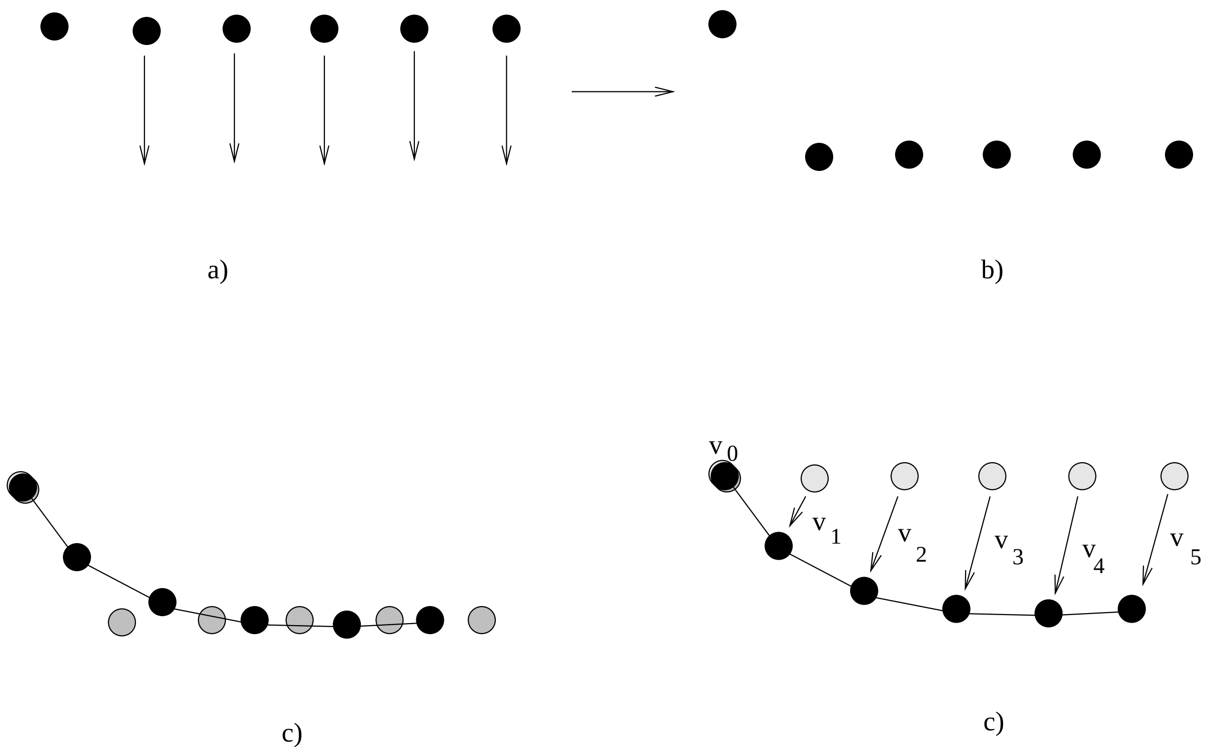


Figure C.1: Example of approximate dynamics system steps applied to a chain of point masses. The initial frame with all of the external forces applied (a). Animating each point mass individually according to their own external forces and initial velocity (b). Restoring the length constraints (c). Calculating the new velocity (d).

```
%t=time interval between frames
%pm=mass of point p
%pnew=new position of point p
%pold=old position of point p
%pv=velocity of point p
%pa=acceleration of point p
For all points, p, that are not fixed in space
    determine external forces, F, on point p
    pa=F/pm
    pnew=pold+pv*t+pa*t*t/2
end for
```

Figure C.2: Pseudocode for animating point masses under external forces.

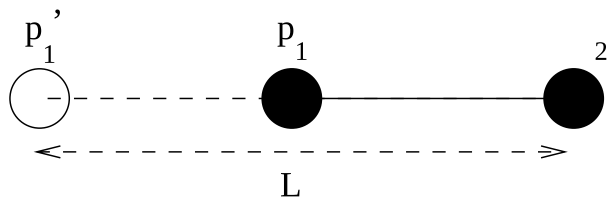


Figure C.3: Example of desired position, $\mathbf{p_1'}$ of point $\mathbf{p_1}$, due to distance from $\mathbf{p_2}$.

C.1 Dynamics Step

The first stage requires applying simple Newtonian mechanics to each point mass. For every point mass we ignore the constraints that attach them to each other in the first stage, so in most cases the only force on each point mass is gravity, F_g. If the point mass penetrates the ground, we apply an elastic force of $k \cdot y$ where k is a spring constant for the force and y is the depth of penetration. The pseudocode for this stage is listed in figure C.2.

C.2 Convergence Step

Now, after the first stage we have a desired position for every point mass. For the point masses that are not fixed, we can calculate other desired positions due to the length constraints. In Figure C.3 we show two point masses $\mathbf{p_1}$ and $\mathbf{p_2}$. A length constraint between the two points that requires the distance between them be L. Since $\mathbf{p_1}$ is closer

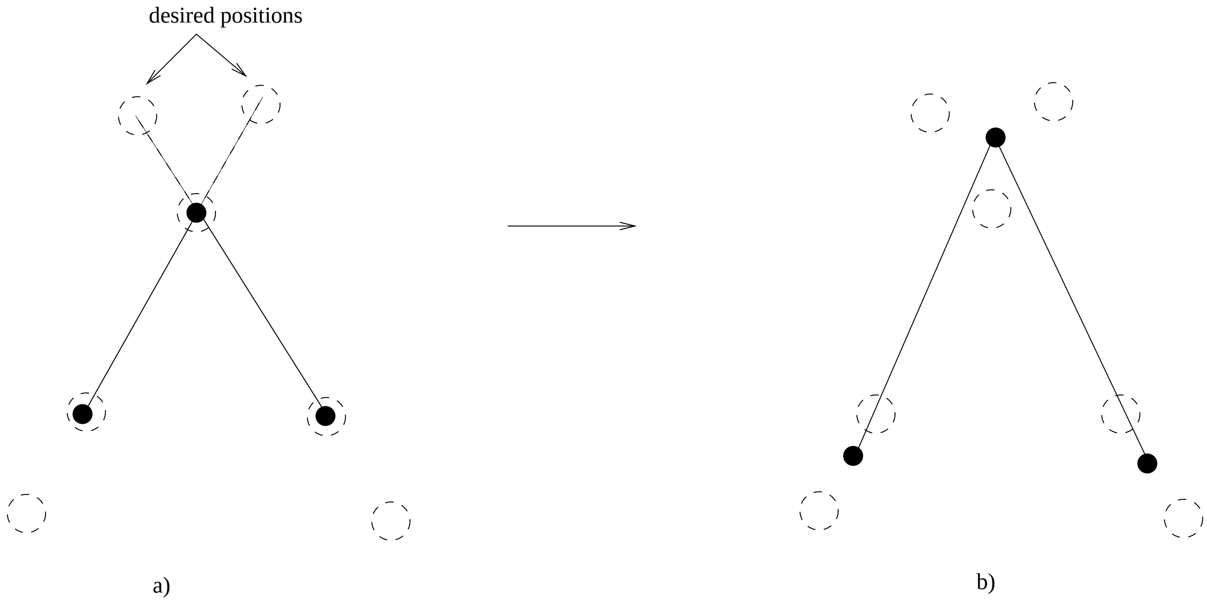


Figure C.4: Example of one iteration of our convergence step, where we apply a weighted average on the desired positions for the points in order to nudge the points to more stable positions

.

than L units away, this would result in an internal force pushing $\mathbf{p_1}$ away from $\mathbf{p_2}$ on the ($\mathbf{p_1}$,$\mathbf{p_2}$) line. Ideally, a desired position for $\mathbf{p_1}$ from $\mathbf{p_2}'s$ standpoint is at $\mathbf{p}_1'$, which we can calculate as

$$\mathbf{p_1}' = \frac{(\mathbf{p_1} - \mathbf{p_2})L}{\|\mathbf{p_1} - \mathbf{p_2}\|}. \tag{C.1}$$

We attempt to restore the length constraints by performing a weighted average on the desired position for every point mass, and then calculating new positions, as shown in Figure C.4. This step is repeated until convergence, or until a limit on the number of iterations has been reached. The weights used for the weighted average are generally equal to the mass of the point mass that the current point is being pushed away from. When the desired position is not the result of length constraints, and instead is the point's current position, we use the point's mass as the weight. The weights could be increased or decreased to alter the strength of various bonds within the structure. In Figure C.5 we show a pseudocode version of the algorithm used in this step.

```
%repeat a few times
for i=0;i<maxIterations;i++

    %calculate new desired position
    for all points p, not fixed
        %add current position
        vect=p*wt(p,p)
        weight=wt(p,p)

        %add desired positions from length constraints
        for all points q that p has a length constraint with
           vect=vect+desiredPos(p,q)
           weight=weight+wt(p,q)
        end for
        tempDesired[p]=vect/weight
    end for

    %update newdesired positions
    for all points p, not fixed
        p=tempDesired[p]
    end for
end for
```

Figure C.5: Algorithm used to restore length constraints between point masses.

C.3 Velocity Calculation

Finally, once the final position for the points has been calculated, we then store the new velocity for the point masses so that we can use them when animating the next frame. For this step, we simply use finite differences. Thus, for an arbitrary point, $\mathbf{p}$, its new velocity, $\mathbf{v}$ is

$$\mathbf{v} = \frac{\mathbf{p} - \mathbf{p}_{\mathrm{old}}}{\triangle t}. \tag{C.2}$$

Appendix D

Procedures for Shoulder and Hip Reconstruction

In this section we outline the functions used to bend and twist the shoulders and hips. The coordinate system is shown in Figure D.1. The procedures involve using the relative position of the feet to calculate the vertical bend, θ, the horizontal bend, ϕ, and the twist, α of the shoulders and hips.

In all of the procedures, we use the following function

$$\text{scaleFcn}(x, scale, min, max, \mathit{offset}) = \begin{cases} \mathit{offset} + \frac{scale}{2} & \text{if } x > max \\ \mathit{offset} - \frac{scale}{2} & \text{if } x < min \\ \mathit{offset} + \frac{scale}{2} \sin(\pi[\frac{(x-min)}{2(max-min)} - \frac{1}{4}]) & \text{otherwise} \end{cases} \tag{D.1}$$

As shown in Figure D.2, scaleFcn is a monotonic function defined between min and max that smoothly tapers off at its lower limit to $\mathit{offset} - \frac{scale}{2}$ and at its upper limit to $\mathit{offset} + \frac{scale}{2}$. We set these limits to equal the limits of the rotation constraints applied on the shoulders and hips.

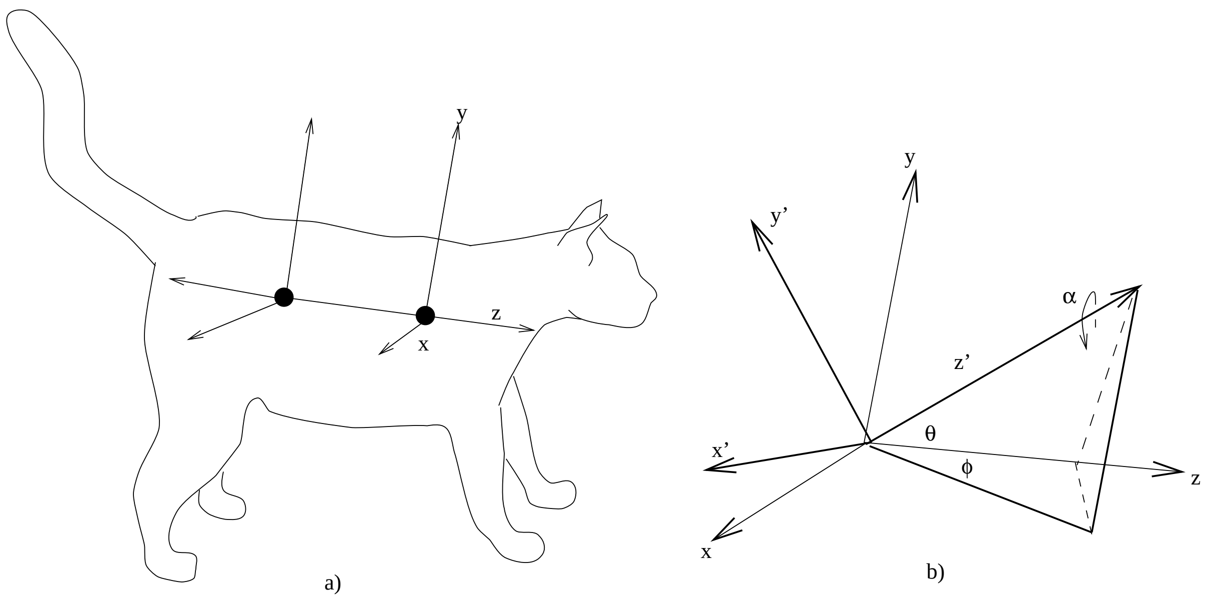


Figure D.1: The model coordinate system for shoulders and hips. (b) The modified coordinate system, along with the three angles which define the modification.

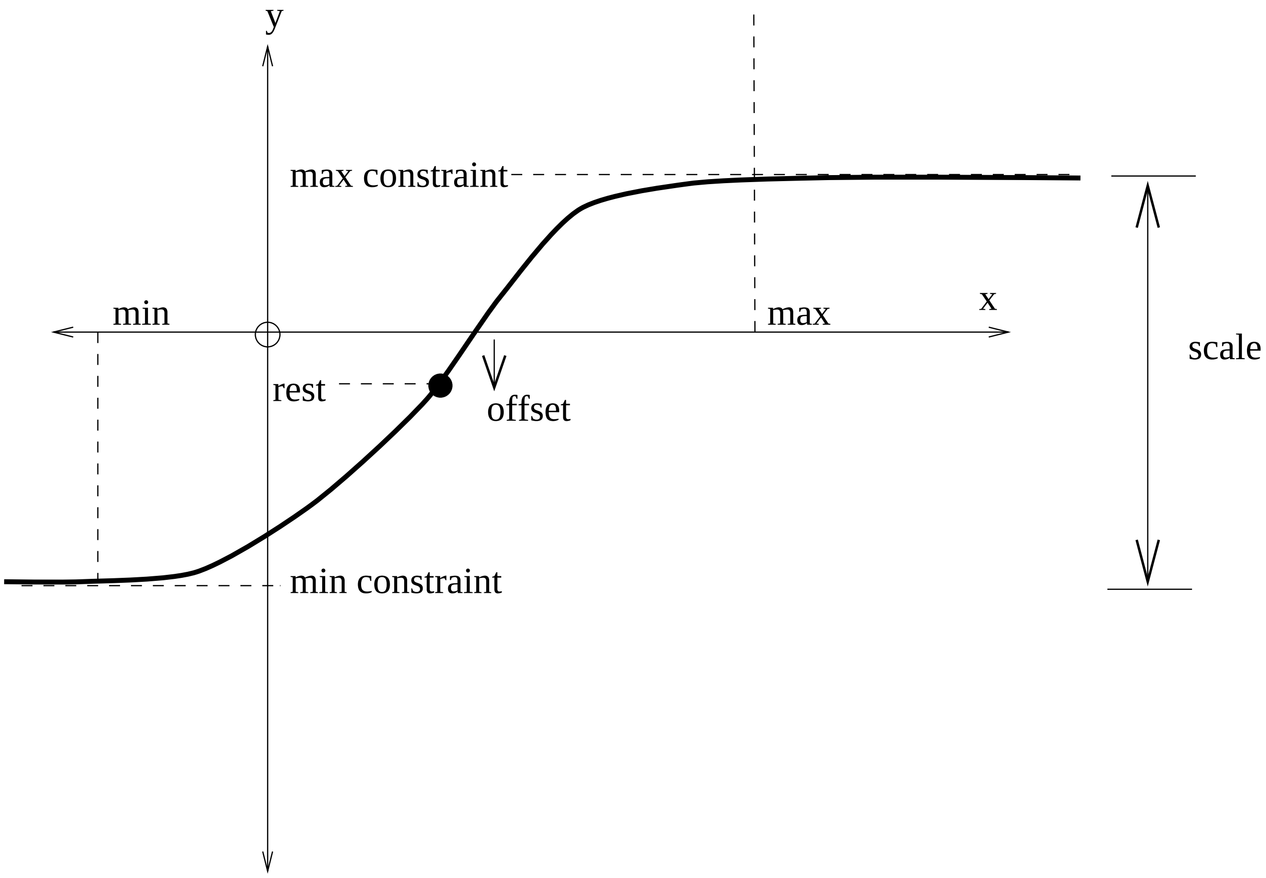


Figure D.2: scaleFcn

D.1 Vertical Bend

The vertical bend parameter is defined in Figure D.1b as θ. It controls how much the spine bends in the yz-plane. We set the bend to be a function of the average y values of the legs relative to the local coordinate system:

$$\theta = \text{scaleFcn}(y_{avg}, \triangle\theta, y_{min}, y_{max}, \theta_{mid}), \tag{D.2}$$

where y_{min} and y_{max} are the limits for the average y values used, $\triangle\theta$ is the difference between the largest allowable value for θ and the smallest, and θ_{mid} is the midpoint between the limits of θ. The result of this function causes the body to bend away from the legs if the average foot height is near the body height, and bend towards the legs if the average foot height is far below the body height.

D.2 Horizontal Bend

The horizontal bend controls how much the creature would bend in its local xz-plane. In Figure D.1(b), we represent this parameter with ϕ. We use the average x values of the legs relative to the local coordinate system to control the horizontal bend. Thus if the legs are close to the left or the right side of the body, the creature will bend in that direction. ϕ is calculated as

$$\phi = \text{scaleFcn}(x_{avg}, \triangle\phi, x_{min}, x_{max}, \phi_{mid}), \tag{D.3}$$

where x_{min} and x_{max} are the limits for the average x values, $\triangle\phi$ is the difference between the limits of ϕ and ϕ_{mid} is the midpoint between the limits of ϕ.

D.3 Twist

With parameter α we rotate the spine segment around the z' axis in Figure D.1(b). In this case, we base the scaleFcn calculation on the relative difference between the y values of the feet. If the left foot is lower than the right foot, we want the body to twist to the left, and if the right foot is lower than the left foot we want the body to twist to the right. Our procedural definition for this is

$$\alpha = \text{scaleFcn}(\triangle y, \triangle\alpha, \triangle y_{min}, \triangle y_{max}, \alpha_{mid}), \tag{D.4}$$

where $\triangle y_{min}$ and $\triangle y_{max}$ are the limits for the $\triangle y$ values, $\triangle\alpha$ is the difference between the limits of α and α_{mid} is the midpoint between the limits of α.

Bibliography

[1] R. McNeill Alexander. *Elastic Mechanisms in Animal Movement.* Cambridge University Press, 1988.

[2] R. McNeill Alexander. *Optima for Animals.* Princeton University Press, 1996.

[3] William W. Armstrong and Mark W. Green. The Dynamics of Articulated Rigid Bodies for Purposes of Animation. In *Graphics Interface '85*, pages 407–415, 1985.

[4] Norman I. Badler, Cary B. Phillips, and Bonnie Lynn Webber. *Simulating Humans: Computer Graphics Animation and Control.* Oxford University Press, 1994.

[5] Norman L. Badler and Cary B. Philips. Interactive Behaviours for Bipedal Articulated Figures. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, July 1991.

[6] Michael Blum and Umakanth Thumrugoti. Dynamics in Feature Animation. In *SIGGRAPH '97 Course Notes (Physically Based Modeling: Principles and Practice)*, pages SG1–SG39. 1997.

[7] Lynne Shapiro Brotman and Arun N. Netravali. Motion Interpolation by Optimal Control. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 309–315, August 1988.

[8] Armin Bruderlin. Dynamics + kinematics = controllable realism. In *SIGGRAPH '90 Course Notes (Human Figure Animation:Approaches and Applications)*, pages 136–144. 1990.

[9] Armin Bruderlin and Thomas W. Calvert. Goal-directed, dynamic animation of human walking. In *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23, pages 233–242, July 1989.

[10] Armin Bruderlin and Tom Calvert. Interactive Animation of Personalized Human Locomotion. In *Graphics Interface '93*, pages 17–23, 1993.

[11] Armin Bruderlin and Lance Williams. Motion Signal Processing. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, volume 29, pages 97–104, 1995.

[12] Byoung S. Choi and Shin Min Song. Fully Automated Obstacle-Crossing Gaits for Walking Machines. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(6):952–964, November/December 1988.

[13] Michael F. Cohen. Interactive Spacetime Control for Animation. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 293–302, 1992.

[14] Teitler Film / Interscope Communications. The Lost World, 1925.

[15] Rebecca L. Craik and Carol A. Oatis. *Gait Analysis*. Mosby, 1995.

[16] Universal Pictures / Amblin Entertainment. Jurassic Park, 1993.

[17] Roy Featherstone. *Robot Dynamics Algorithms*. Kluwer, 1987.

[18] M. Girard and A. A. Maciejewski. Computational Modeling for the Computer Animation of Legged Figures. In *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 263–270, July 1985.

[19] Michael Girard. Interactive Design of 3D Computer-Animated legged animal motion. *IEEE Computer Graphics and Applications*, 7(6):39–51, June 1987.

[20] Michael Girard. US Patent 5594856: Computer User Interface for Step-driven Character Animation, 1997.

[21] James Gray. *Animal Locomotion*. Weidenfeld and Nicolson, 1968.

[22] Larry Gritz and James K. Hahn. Genetic Programming for Articulated Figure Motion. *Journal of Visualization and Computer Animation*, 6(3):129–142, July 1995.

[23] Jack Hamm. *How to Draw Animals.* Perigee Books, 1982.

[24] Jessica K. Hodgins and Marc H. Raibert. Adjusting Step Length for Rough Terrain Locomotion. *IEEE Transactions on Robotics and Automation*, 7(3):289–298, June 1991.

[25] Jessica K. Hodgins, Paula K. Sweeney, and David G. Lawrence. Generating Natural-looking motion for Computer Animation. In *Graphics Interface '92*, pages 265–272, 1992.

[26] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating Human Athletics. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, volume 29, pages 71–78, 1995.

[27] Verne T. Inman, Henry J. Ralston, and Frank Todd. Human locomotion. In Jessica Rose and James G. Gamble, editors, *Human Walking*, pages 1–22. Williams and Wilkins, 1994.

[28] David Kahaner, Cleve Moler, and Stephen Nash. *Numerical Methods and Software.* Prentice Hall, 1989.

[29] Brian Knep, Craig Hayes, Rick Sayre, and Tom Williams. Dinosaur Input Device. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI '95)*. ACM Press, May 1995.

[30] Hyeongseok Ko and Norman I. Badler. Straight Line Walking Animation. In *Graphics Interface '92*, pages 273–281, 1992.

[31] Evangelos Kokkevis, Dimitri Metaxas, and Norman I. Badler. Autonomous Animation and Control of Four-Legged Animals. In *Graphics Interface '95*, pages 10–17, 1995.

[32] Joe F. Lazlo. Controlling Bipedal Locomotion for Computer Animation. Master's thesis, University of Toronto, 1996.

[33] Philip Lee, Susanna Wei, Jianmin Zhao, and Normal I. Badler. Strength Guided Motion. *Computer Graphics*, 24(4):253–262, 1990.

[34] Zicheng Liu and Michael F. Cohen. Decomposition of Linked Figure Motion Diving. In *Eurographics '94 (Fifth Eurographics Workshop on Animation and Simulation)*, 1994.

[35] Zicheng Liu and Michael F. Cohen. Keyframe Motion Optimization By Relaxing Speed and Timing. In *Computer Animation and Simulation '95*, pages 144–153, 1995.

[36] Zicheng Liu, Steven Gortler, and Michael F. Cohen. Hierarchical Spacetime Control. *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 35–42, July 1994.

[37] Winsor McCay. Gertie the dinosaur, 1914.

[38] Eadweard Muybridge. *Animal Locomotion*. Dover Publications, 1979.

[39] John Nagle. Realistic Animation of Legged Running on Rough Terrain. In *Proceedings of the 1995 IEEE Computer Animation Conference*, April 1995.

[40] J. Thomas Ngo and Joe Marks. Spacetime constraints revisited. *Computer Graphics (SIGGRAPH '93 Proceedings)*, 27:343–350, 1993.

[41] LucasArts Entertainment Company / Nintendo of America Inc. Star Wars: Shadows of the Empire, 1996.

[42] Frederic I. Parke and Keith Waters. *Computer Facial Animation*. A. K. Peters Ltd., 1996.

[43] Ken Perlin. Danse Interactif. In *SIGGRAPH '94 Electronic Theatre*, 1994.

[44] Ken Perlin. Real Time Responsive Animation with Personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, March 1995.

[45] First National Pictures. The Lost World, 1925.

[46] MCA Television Entertainment Inc. / Renaissance Pictures. Hercules: The Legendary Journeys, 1995.

[47] Universal Pictures. Dragonheart, 1996.

[48] Unreal Pictures/Kinetix. BIPED Character Studio: Two-legged Animation, 1997.

[49] Xavier Pintado and Eugene Fiume. GRAFIELDS: Field-directed Dynamic Splines for Interactive Motion Control. *Computers and Graphics*, 13(1):77–82, 1989.

[50] Marc H. Raibert. *Legged Robots that Balance*. MIT Press, 1986.

[51] Marc H. Raibert and Jessica K. Hodgins. Animation of Dynamic Legged Locomotion. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 349–358, 1991.

[52] Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. Efficient Generation of Motion Transitions using Spacetime Constraints. In *Computer Graphics (SIGGRAPH '96 Proceedings)*, volume 30, pages 147–154, 1996.

[53] Robert E. Rosenblum, Wayne E. Carlson, and Edwin Tripp III. Simulating the Structure and Dynamics of Human Hair: Modelling, Rendering and Animation. *The Journal of Visualization and Computer Animation*, 2:141–148, 1991.

[54] Karl Sims. Evolving virtual creatures. *Computer Graphics (SIGGRAPH '94 Proceedings)*, 28:15–34, 1994.

[55] A. James Stewart and James F. Cremer. Algorithmic Approach to Animation. In *Graphics Interface '92*, pages 273–281, 1992.

[56] David Sturman. Interactive Keyframe Animation of 3D Articulated Models. In *Graphics Interface '84*, pages 35–40, 1984.

[57] Munetoshi Unuma, Ken Anjyo, and Ryozo Takeuchi. Fourier Principles for Emotion-based Human Figure Animation. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, volume 29, pages 91–96, 1995.

[58] Michiel van de Panne and Eugene Fiume. Sensor-Actuator Networks. In *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 335–342, 1993.

[59] Michiel van de Panne, Ryan Kim, and Eugene Fiume. Virtual Wind-up Toys for Animation. In *Graphics Interface '94*, pages 208–215, 1994.

[60] Michiel van de Panne and Alexis Lamouret. Guided optimization for balanced locomotion. In *Computer Animation and Simulation '95*, pages 165–177, 1995.

[61] Michiel van de Panne and Alexis Lamouret. Motion Synthesis By Example. In *Computer Animation and Simulation '96*, pages 199–212, 1996.

[62] Cornelius W. A. M. van Overveld. An Iterative Approach to Dynamic Simulation of 3-D Rigid-Body Motions for Real-Time Interactive Computer Animation. *The Visual Computer*, 7:29–38, 1991.

[63] Cornelius W. A. M. van Overveld and Hyeongseok Ko. Small Steps For Mankind: Towards a Kinematically Driven Dynamics Simulation of Curved Path Walking. *The Journal of Visualization and Computer Animation*, 5:143–165, 1994.

[64] Chris Welman. Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation. Master's thesis, Simon Frasier University, 1993.

[65] Jane Wihelms and Brian A. Barsky. Using Dynamics Analysis to Animate Articulated Bodies such as Humans and Robots. In *Graphics Interface '85*, pages 97–104, 1985.

[66] Andrew Witkin and Michael Kass. Spacetime Constraints. In *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 159–168, 1988.

[67] Andrew Witkin and Zoran Popovic. Motion Warping. In *Computer Graphics (SIGGRAPH '95 Proceedings)*, volume 29, pages 105–108, 1995.