Computational Generation of Terrestrial Animal Locomotion

Kevin Wampler

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2012

Reading Committee:

Zoran Popović, Chair

Jovan Popović

Emanuel Todorov

Program Authorized to Offer Degree:
UW Computer Science and Engineering

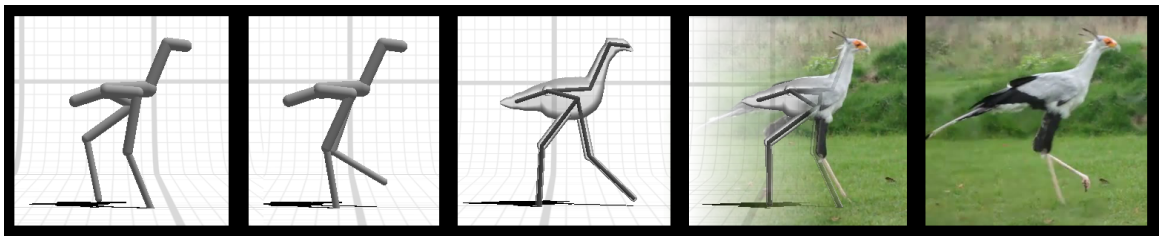
University of Washington

Abstract

Computational Generation of Terrestrial Animal Locomotion

Kevin Wampler

Chair of the Supervisory Committee:
Professor Zoran Popović
UW Computer Science and Engineering



While the animation of humans is a well established topic in computer graphics, the animation of animals remains far less explored. Part of this is because in contrast to the wealth of easily obtainable motion data for humans, similar motion data for animals or fictional creatures is substantially more difficult or impossible to obtain. In this thesis I develop techniques for animating animals which do not rely on any motion capture data, but instead automatically synthesize motions by exploiting principles from physics and biomechanics.

I address three main components to the problem of animal motion synthesis. First, I present an optimization technique which is capable of solving for a cyclic gait for an animal without requiring any motion data, and which can additionally solve for the most efficient type of gait for the animal to perform as well as refine the shape of the animal itself if desired. Second, I address the problem of increasing the realism of the synthesized gaits by learning from a database of ground-truth animal gaits. Finally, I move beyond gait synthesis to the synthesis of kinematic controllers which can be interactively guided by a user to create animations in real-time.

TABLE OF CONTENTS

Page

LIST OF FIGURES

ACKNOWLEDGMENTS

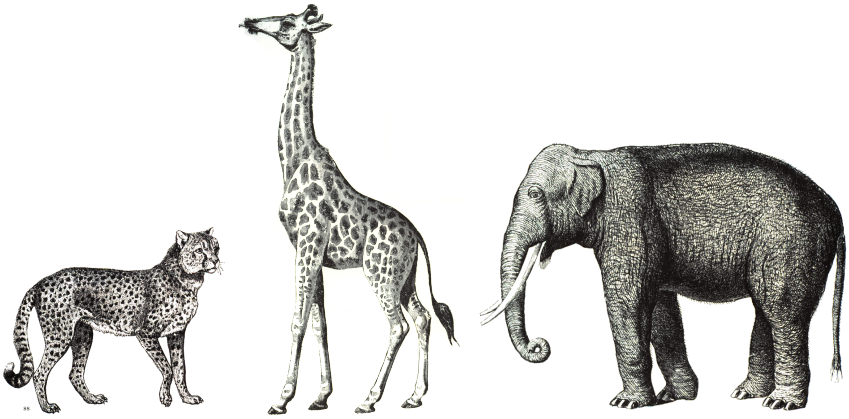
I would like to thank my advisors Zoran Popović and Jovan Popović who have collaborated with me in the projects described in chapters 3-5 and who have provided ideas and insights related to my research which are too numerous to mention. Additionally, I would like to thank the collaborators on my research papers done during my time at the University of Washington which are not covered in this thesis [77, 46, 78]: Yongjoon Lee, Erik Andersen, Evan Herbst, Gilbert Bernstein, Daichi Sasaki, and Li Zhang. I am also greatly appreciative of the input of Adrien Treuille during the formative stages of the project described in chapter 3. Finally, I would like to thank my fiancée Lillie Kittredge who helped to keep me sane and subsisting on more than Pop-Tarts during some of the more intense work crunches involved in the projects appearing in this thesis.

Chapter 1

INTRODUCTION

Consider an animal, perhaps a small and agile one such as the four-horned antelope:

Of course the above is only a single static picture of the animal, but in your mind's eye you can well imagine how it might move. A moment of reflection about this animal and the motion you envisioned for it will likely reveal that the nature of the motion was well matched to the specific shape of the animal in question. If you picture motions for animals shaped differently than the above four-horned antelope:



Then it is likely that the qualitative nature of these motions will be quite different. This much is relatively obvious – that the form of an animal is intertwined with the types of

Figure 1.1: Top: paintings from the Lascaux caves in France. Bottom: portion of a horse motion sequence by Eadweard Muybridge

motions that the animal will likely make. But it is far less clear what the precise nature of this relationship is, and how computational tools can be developed to exploit it.

This question of the relationship between the form and motion of animals has, in one guise or another, an extensive and long history. Both scientists and artists have long been fascinated by the wealth of forms exhibited by the animals found in nature and by the varied and elegant ways in which many of these animals move. One can find depictions of animals in motion back into the Paleolithic era (1.1, top), and continue through the origins of cinematography (1.1, bottom) to modern computer animation techniques.

Modern technology provides some very powerful tools for studying and authoring the motion of animals. In particular, if the animal in question happens to be human, then it is especially simple to determine how it should move. One need simply to hire an actor and

instruct them to perform the desired motion. This performance can then be captured on video, or even better in full 3D by using relatively widespread and well-developed motion capture technology. This puts a wealth of high-quality data at the fingertips of anyone wishing to either study or animate the motion of humans.

Unfortunately such rich data for the motion of non-human animals is at best much harder to obtain, and in the case of extinct of fictional animals data of this sort is completely nonexistent. This difficulty in obtaining data on the motion of animals has of course not diminished the interest in their study, as the motion of animals presents a fascinating topic. The shapes and motions of animals vary widely, exhibiting a vast and ingenious variety of adaptations to different needs and circumstances. Furthermore, this variation is far from random, but rather the motion of an animal is closely tied to its shape and to the evolutionary niche in which it must survive. We would of course not expect an elephant to move in the same way as a mouse, or for an animal shaped like a cheetah to only move like a tortoise. Almost anywhere in nature one looks, the form of animals is intimately tied to the functions that their form allows.

This clear relationship between form and function seen in the motions of many animals points to the possibility of using biomechanical principles to generate and study these motions. It is then perhaps little surprise that one of the earliest pre-scientific texts on biomechanics as well as the earliest properly biomechanical work both bear the title *De Motu Animalium* (On the Motion of Animals, by Aristotle and Borelli respectively) [3, 6]. This raises the possibility that, at least in some cases, it may be possible to accurately estimate the motion of an animal given only information about its shape. If this process can be automated, then it provides an easy way to circumvent the difficulty in acquiring high-quality data for the motion of an animal – the data can simply be generated from scratch. Furthermore it would allow reproduction of the motion of extinct creatures for which there is no information possible about their motion other than that dictated by their shape (and possibly some preserved sets of footprints).

In this thesis I will consider the problem of automatically determining how an input animal might locomote. I will focus in particular on legged locomotion, and put particular focus on the synthesis of cyclic gaits as in some sense these represent the 'core' of how an

animal locmotes. In the remainder of this chapter I will describe the overall approach I take to animal motion synthesis at a high level. The next chapter will then provide some overall context to computational approaches to motion synthesis. After this, I will describe the methods I have developed in three stages. First I will focus on the problem of generating a motion at all, as this turns out to be a difficult computational problem. Next I will consider how the realism of the synthesized motions can be improved, and finally I will cover the synthesis of non-cyclic locomotion. Finally, I will conclude by looking at possible directions for future research in this area.

1.1 Optimality in Motion

While it is intuitively clear that the form of an animal informs its motion, merely noting this interrelationship is of little help when it comes to actually determining how a particular animal should move. In order to actually create algorithms which can assist in animating an animal without recourse to motion data, it is necessary to mathematically specify what the nature of this interrelationship is. Unfortunately we might in general expect the reasons for a particular species of animal's motion to be a complicated combination of the circumstances of the animal's evolutionary niche and genetic heritage, a complete description of all the nuances of which would be impractical. Nevertheless, for many practical applications it is not necessary that all the nuances of the relationship between form and motion be explained in full and exact detail. Instead, a great deal of progress can be made by capturing the most important broad principles governing an animal's motion.

Fortunately, there are many circumstances in which a simple unifying principle can go a great ways toward describing why an animal moves in the way that it does. This principle, that of *optimality*, was studied in the context of animal motion at least as early as the nineteenth century:

> In the course of my investigations, I have met with numerous instances, in the muscular mechanism of the vertebrate animals, of the application of the principle of least action in Nature ; by which I mean that the work to be done is effected by means of the existing arrangement of the muscles, bones, and joints, with a less expenditure of force than would be possible under any other

> arrangement ; so that any alteration would be a positive disadvantage to the animal.
>
> (*Principles of animal mechanics* v–vi, *Samuel Haughton* 1873 [25])

Ironically, today Samuel Haughton is often remembered for his blistering critique of Darwin, whereas now the same principle of optimality would usually be justified on evolutionary grounds – in many circumstances the fitness of a terrestrial species is improved by being able to locomote without an excessive waste of energy [1].

This principle of the optimality in an animal's motion is the primary conceptual and algorithmic tool which I exploit throughout this thesis. Although such optimality admits a particularly clean and general mathematical framework, it is useful to first see some 'real-world' instances where this principle may be seen at work.

Examples of this principle of optimality in locomotion can be seen everywhere in nature. Many simple examples can be reasoned about intuitively. For example, one would expect heavier animals to move in ways which minimize excessive torques on their joints as well as lateral forces on their bones, and indeed it can be observed that many of the heaviest land animals tend to walk with relatively stiff and straight legs and tend to avoid "bounding" type gaits (see figure 1.2). Numerous other similar examples can be brought to mind, for instance how the gaits of slow versus fast moving animals differ, or how high an animal tends to lift its legs off the ground in different environments (for instance in shallow water verses dry land).

Although these preceding examples are largely intuitive, more rigorous studies of animal locomotion have been performed by many authors. For instance humans (along with other animals) tend to increase the frequency of their gait as their speed changes, taking quicker steps at higher speeds and slower steps at lower speeds. For humans, this relationship has been studied in some detail, and can be experimentally studied by instructing a subject to walk on a treadmill in time with a metronome and measuring their rate of oxygen consumption. Not surprisingly the observed relationship appears to be well explained by optimality [5], with the stride frequency voluntary chosen by humans (i.e. without a metronome) falling near the metabolically optimal one [92, 1].

Another well-studied example can be observed in how different gaits are chosen by an

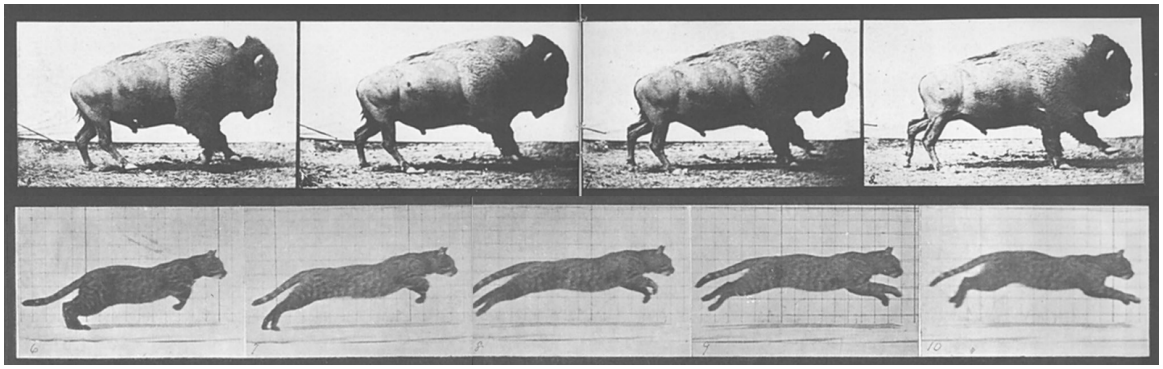


Figure 1.2: A comparison of the galloping motion of a buffalo and a cat, taken from *Animals in Motion* by Eadweard Muybridge [59]. The difference in mass and shape between the two animals is clearly reflected in their motions.

animal. In bipeds, including humans, an animal will tend to use a walking gait at low speeds, and abruptly transition into a running gait at higher speeds. This phenomenon is also well explained as emerging from the requirement that the animal move using minimal (or near-minimal) metabolic energy. At lower speeds walking gaits are more efficient because they lack a flight phase where both feet leave the ground, and thus the animal can avoid the expenditure of energy required to perform a 'jump' each gait cycle. However at higher speeds a flight phase allows for a higher velocity without excessively long or fast strides.

The purpose of these examples is not to provide an exhaustive survey of research into the use of optimality in studying animal locomotion, but rather to provide a few illustrative examples of its use. Of course, knowing that animal motions can be well described by the property of being optimal does not directly tell us how a given animal should move. This is both because it is not always immediately obvious what the animal should be optimal with respect to, and because even when the optimality criteria are fully specified it can still be difficult to solve for the motion which satisfies these criteria.

Solving for how an animal should move based on the motion's optimality is particularly difficult when one desires a complete motion which is fit to serve as an animation, rather than just a model of some simpler abstracted property such as just the stride frequency, type of gait, etc. This is because when one only cares about some high-level feature of a motion, it is

often possible to use a highly simplified model of the physical characteristics of the animal, whereas a fully described motion generally requires a physical model with substantially greater complexity. Within some sufficiently simple physical models the optimal motion can be solved for analytically, but even when an analytic solution is impossible, basic numeric optimization techniques are often effective on simplified models. Unfortunately, the space of possible motions is vastly larger for a full physical model of an animal, and the process of searching for optimal motions within these models is complicated in numerous ways relative to what is required for a simplified model. In addition, simple models admit simpler descriptions of what it means to be optimal, and optimality criteria which are perfectly sufficient to glean insights from a simple model may still lead to incorrect looking motions when used on a full physical model.

Fortunately, techniques have been developed which can sometimes solve for the full motion of a character (normally a human) by solving an optimization problem. This approach has been well studied in computer graphics, where the applications generally require a complete animation, and thus just knowing some simplified abstraction such as the stride frequency is of limited use. Unfortunately, however, despite a great deal of study these approaches remain plagued by a host of practical difficulties. This has traditionally limited their usage to either very simple characters or to situations where some data of the character performing a similar motion to the desired one can be obtained. This makes these techniques generally unsuitable for animating animals, since this data is very difficult to obtain. Thus there remain many practical difficulties which must be overcome in order to actually achieve the goal of being able to accurately recreate and animate the motions of animals living, extinct, and fictional.

In the remainder of the paper I will outline three projects which aim to allow realistic motions for an animal to be synthesized from only a small amount of easily obtainable data. In particular, each proposed project will describe a technique which only requires as its input a description of the shape on the animal, the masses of its limbs, the allowed angular rotations of its joints, and a small amount of other information such as some basic information about the environment in which the animal is moving or what sorts of motions it should be able to perform. From this input the proposed methods will automatically

synthesize the required motions for the animal.

All three methods are based on constructing optimal motions. Because solving for such motions is by itself a significant challenge, the first project will involve synthesizing a gait cycle for a given input animal such that the resulting motion is both physically realistic and at least relatively close to optimal. For this first project the definition of 'optimal' will be taken for granted, and modeled by hand as a combination of simple biologically-motivated terms, such as minimizing joint torques and keeping the head stable. In reality however, the correct definition of optimality is itself a problem without an obvious answer, so the second proposed project will revolve around finding the correct definition of 'optimal' so that the resulting motions best match those actually exhibited by animals, and extrapolating this notion of optimality to animals for which no motion data exists. Finally, the previous two projects only involve synthesizing cyclic gaits, so the final proposal will describe how an entire vocabulary of interconnect motions may be synthesized so that an animal can use these motions to optimally accomplish a range of user-specified tasks.

Chapter 2

OVERVIEW

The animation of virtual characters, whether animal, human, or mechanical, has a received an extensive amount of study from the computer graphics community. It comes as no surprise then that a wide range of different approaches for creating character animations have been developed over the years. Although the details of these techniques vary widely, there are some common assumptions and representations shared by the vast majority of approaches. These representations underlie both non-optimization-based animation techniques as well as optimization-based techniques such as those on which my work is based.

In this chapter I'll start by providing some background common to most character animation techniques. After this I will provide an overview of related work in character animation, although more focused descriptions of the related literature will also be provided when relevant in the later chapters in this thesis. Finally I will give a high-level overview of how optimization can be employed to synthesize motions in computer graphics.

In animating an animal or other character, the first choice which needs to be made is how the character itself is represented. For most computer animation applications, this representation consists of two parts: one of which is well-suited for representing a high-quality shape for the character, and another which is well-suited to representing high-quality motion. In this thesis I am only concerned with a character's motion, and thus will largely ignore how the character's detailed shape is represented. Still, since the character animations one typically sees combine both the visual and motion representations, I'll provide a brief (and somewhat simplified) description of how this combination is achieved so that these animations can better be placed in context.

The visual shape of an animated character is typically represented as a surface described by a mesh of connected flat triangular patches, known as a *triangular mesh*. This representation is highly flexible, and by increasing the number of triangles in such a mesh, models

with a very high degree of visual fidelity can be created. Unfortunately, the high number of triangular components in such a mesh means it would be a complicated affair to animate a mesh by directly describing the motion of each triangle.

In order to animate a character, a secondary representation called a *skeleton* is typically used. Although a skeleton in the sense used in animation is different from its biological counterpart, the name is suggestive of some similarities. Many of the bones in a real-world skeleton serve to move large chunks of a creature essentially together as a single unit. For instance the radius and ulna in a human constrain most of the forearm to move as "one chunk", with the only significant deformations occurring around joints [1] (such as the wrist or elbow in this case). A skeleton in animation is used in a similar way. In a process known as *rigging* each of the triangles in the mesh defining a character's shape is associated with a bone in the character's skeleton, or possibly multiple bones in the case of triangles lying near joints. The motion of each triangle is then automatically derived from the motion of the bones to which it is attached. This allows an animator to simply specify the motion of the characters skeleton, and from this, a motion for the character's full high-quality mesh is automatically obtained.

Since I am concerned with the the motion of animals in this thesis, I will focus only on the skeletal description of a character, since describing an animation for such a skeleton is sufficient to specify the animation for a full high-quality visual character. As alluded to earlier, this viewpoint has a great advantage in animation, as an animation for a skeleton can be described in a relatively simple and compact form (which is, after all, why they're generally used in the first place).

The first step to describing a full animation is describing the configuration of a skeleton at a single instant in time. Such a configuration is called a *pose*, and essentially describes the position and orientation of each of the bones in a skeleton. Not all potential configurations of a skeleton's bones represent valid poses, however, as the relative positions and orientations of certain pairs of bones are restricted when the bones are connected by a *joint*. As a simple example, the valid poses of a human-like character are restricted to those where the upper

[1] As a contrast, consider the motion of something like an amoeba for which a skeletal description is poorly suited.

arm actually connects to the forearm, rather than allowing the upper and forearms to move completely independently.

Although describing the valid poses for a completely general skeleton can be a somewhat involved ordeal, the skeletons of most real-world animals are fortunately of a particular form where the valid poses may be described in a simple manner. The skeletal descriptions appropriate for animating most real-world animals fit a particular form known as a *tree*. That is, if one traces out the connections between the bones in the skeleton, one finds that there are no connected loops. If one picks a particular joint in a skeleton to start from, called the *root*, such a skeleton has the property that each bone can be reached in exactly one way by traversing a path of connected bones and joints starting from the root.

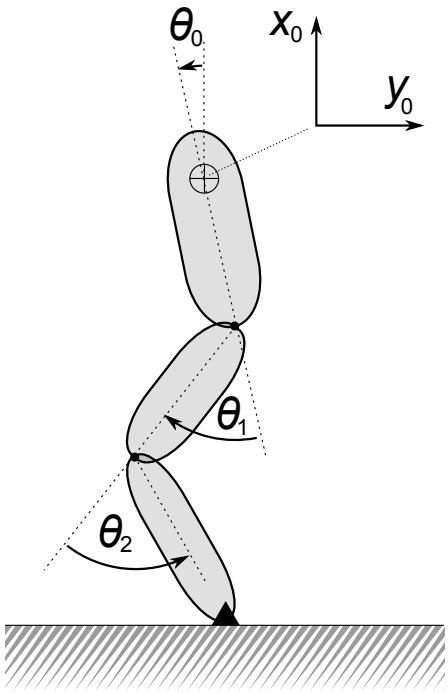


Figure 2.1: A simple 2D monopedal hopper serving as a basic example of a animatable character. This hopper has three limbs connected by two joints, and a single foot (black triangle). A pose for this hopper can be described by a five-dimensional vector $[x_0, y_0, \theta_0, \theta_1, \theta_2,]^T$ where $[x_0, y_0]^T$ gives the translation of the character's root, θ_0 gives the rotation of the root, and θ_1, θ_2 give the rotations at each of the character's two joints.

A simple illustrative example of such a skeleton is shown in figure 2.1. Each joint connects a pair of bones: a *parent* bone which is closer to the root and a *child* bone which is further from the root. A joint then describes the transformation from the parent bone to the child bone. In most cases this transformation will be a simple rotation, and the angle of this rotation can be described with a single parameter in two dimensions, and between one and three parameters in three dimensions. Because there is a unique path from the root of a skeleton to each bone, the position of each bone can be uniquely specified by describing the position and orientation of the root as well as the transformations of each joint along the path from the root to the bone.

A pose for a skeleton can then be given by a vector of numbers describing the position and orientation of the skeleton's root as well as the transformations of each of a skeleton's joints. Throughout this thesis I'll use the term "pose" to refer to this vector of numbers as well as the configurations of a character's bones which it induces, the distinction being clear from context. This description is relatively compact, and allows the pose of a typical skeleton to be described with a vector with a typical size of between five and forty values, depending on the animal. Some more detail on the mathematical specifications of these transformations is given in appendix A, but the high-level description given so far should be sufficient to get a good intuition for the matter.

Of course, as this thesis is concerned with the motion of animals it is necessary to go beyond the description of static poses to the description of a complete motion. In computer animation this is typically achieved in roughly the same manner as traditional representation of motion, such as cell animation or movie reels. To represent a motion, time is discretized into a number of discrete steps called *frames*, and the character's pose is specified for each frame. Provided the time interval between frames is sufficiently small (on the order of $\frac{1}{30}$th of a second), the poses within each frame can then be rapidly replayed to convey a sense of continuous movement. An entire motion can then be represented by a sequence of vectors, each of which describes the character's pose at a single frame.

In principle an animation for a character could be described by directly authoring the poses for the character at each frame. While possible, this method takes a great deal of artistic expertise as well as a significant amount of effort to perform well. Therefore research

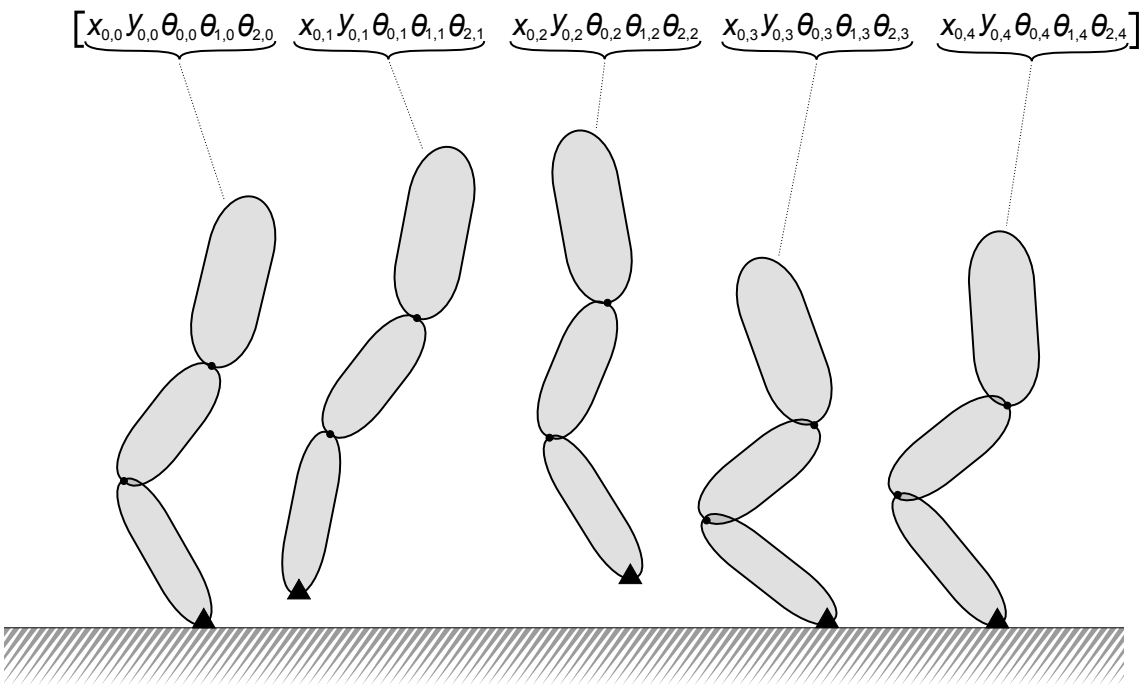


Figure 2.2: An example of a simple five-frame animation for a 2D monopedal hopper. Each frame is parameterized by a five-dimensional giving a single pose for this hopper, $[x_{0,i}, y_{0,i}, \theta_{0,i}, \theta_{1,i}, \theta_{2,i},]^T$, as described in figure 2.1. The entire motion can therefore be parameterized by a 25-element vector made by concatenating the vectors for each individual frame.

in computer animation, including that presented in this thesis, focuses on providing higher-level tools to create these animations. Although my research is concerned with automatically animating animals by leveraging some principles from biomechanics, it's useful to get an idea of the variety of techniques which have been developed for solving similar sorts of problems. I'll give an overview of some of these approaches next.

2.1 Related Work

The animation of virtual characters is a broad and well-studied field within computer graphics. Indeed, the field is large enough to have several different subfields which are often treated independently, for instance a substantial number of papers could be found devoted to each of locomotion, face animation, clothing animation, non-rigid deformations, object manipu-

lation, etc., to say nothing of the enormous range of computer vision papers dedicated to reconstructing character motion from recorded data. While all of these subfields pertain to important aspects of character motion, I am primarily concerned with locomotion within this thesis. This is an especially useful area to focus on in the context of animal animation, as differences in locomotion styles are one of the most obvious contexts in which the differing forms of various animals leads to different motions.

By *locomotion* I am referring in particular to legged terrestrial locomotion, which is a process by which terrestrial animals use their feet in conjunction with contacts with the ground to move in a desired manner. Real-world examples of different types of locomotion include the walks, hops, and runs exhibited by bipeds, the walks, trots, paces, canters and gallops employed by quadrupeds, as well as a few more exotic examples such as a kangaroo's use of its tail as a 'fifth leg' during some walking motions. Needless to say, the examples of potential motions for fictional animals are similarly varied. While not all terrestrial animals use legged locomotion (for instance snakes), nor do all the motions of legged animals constitute locomotion, it nevertheless forms the core mechanism by which most terrestrial animals move about, and has correspondingly received a great deal of attention in both computer graphics and biomechanics.

In everyday life locomotion is an activity that we see and do without too much regard. This commonplace nature of locomotion can mask the difficulties involved in simulating it. It is often the case, however, that some of the most difficult things to program a computer to do well are things which we ourselves find to be second nature, and locomotion seems to be no exception to this. This helps to explain the large array of different approach which have been developed to aid in creating locomotion for virtual characters. These approaches span a spectrum from directly recording human performances, to helping artists create animations, to synthesizing animations automatically.

The easiest way to get high-quality animation for a virtual character, particularly a human character, is to simply record the performance of a real-world actor. The process of recording such a performance and transferring the recorded motion into a sequence of poses for a virtual character is called *motion capture*. The difficulty in motion capture is that it is ambiguous, or at best complicated, to determine how exactly an actor in moving

in 3D from only 2D video recordings. Although this difficulty has kept motion capture from advancing to what might be considered a completely solved technology (at least on a budget within the means of an average person), numerous approaches have been developed which can often give high-quality results [83, 55, 54].

Despite the wide range of motion capture techniques available, the standard method for capturing high-quality motion capture data, particularly in industry, remains *optical motion capture* [86]. In optical motion capture an actor is outfitted with a set of specialized markers designed to facilitate computer algorithms in reconstructing the actor's motion from video. These markers generally take the form of either small retroreflective patches/spheres, or of light-emitting diodes (LEDs). These markers can be relatively easily tracked from 2D video data, and are therefore placed on the actor at locations for which accurate tracking is important, such as the head, torso, hands, feet, knees, and elbows. An actor wearing these markers can then perform a motion in a space equipped with multiple simultaneously recording high-resolution video cameras. This yields a series of synchronized 2D videos from which the 3D location of each marker can be determined by triangulation. Finally, the series of poses corresponding to the actor's motion are fit to this sequence of 3D marker positions.

This method of optical motion capture has the ability to provide very high quality animations and has a number of relatively mature commercial implementations, such as Vicon (`www.vicon.com`) and Qualisys (`www.qualisys.com`). Unfortunately these approaches have some shortcomings which are particularly pronounced in the case of capturing animal motion. Standard optical motion capture methods not only require expensive indoor setups costing tens to hundreds of thousands of dollars [67], but the requirement that the motion be filmed simultaneously from a number of different angles necessitates that an animal be outfitted with a set of markers and made to perform any desired motions within a relatively small space. This introduces a host of practical complications such as the need for a skilled trainer, specialized methods of affixing the markers, transportation of the animal, etc [50, 60]. Although there are companies which specialize in motion capture for animals, the aforementioned difficulties mean that such data is relatively rare even for common animals, and is often completely unavailable for less common animals. Furthermore, even if these

difficulties were to be alleviated, such motion capture techniques would remain inapplicable to extinct or fictional animals.

When acquiring motion capture for an animal is impractical or impossible, the most common alternative used in practice in to have an artist design the desired motion by hand. This approach has the obvious advantage that it requires no access to expensive facilities and no animal training, nor does the animal to be animated need to be easy to acquire or even exist at all.

Unfortunately the simplicity and flexibility of artist-designed animal locomotion comes at a rather obvious cost: the skill and effort needed to author the animations. In particular a great deal of skill is required to generate convincing motions without using reference video footage. The most popular method for artist authoring for skeletal animations is *keyframe animation* where the artist manually specifies the pose of the character at a subset of the frames in an animation, and then the remaining 'in between' frames are automatically interpolated to complete the animation. Designing a realistic animation in this manner requires a practiced eye for how the individual keyframes will look when played in sequence in the final animation. Although numerous approaches have been developed to assist the artist in designing and editing animations [31, 39, 33, 70, 32] the process of designing a realistic motion for an animal 'by hand' is time-consuming for experts and impractical for novices.

2.2 Optimal Motion Synthesis

There is a particularly powerful and appealing way of generating motions which is well suited to animal locomotion. The overall framework to be employed goes something like this: first carefully define which sorts of motions are "better" than others, then computationally search for the "best" possible motion, possibly subject to other constraints. I will refer to such methods as *optimization* methods for animal locomotion.

There are several advantages to adopting such a viewpoint. The first is that it frees one from having to describe the fine-level moment to moment aspects of a motion, instead focusing on much higher-level aspects of the motion. For instance, in an optimization approach one might specify that a motion be smooth, or that it be energetically efficient,

leaving the details of how to exactly achieve this to be determined by the optimization process. This greatly simplifies the animation process, and as shown in this thesis often allows it to be automated completely in the case of animal locomotion.

A potentially more important aspect which makes optimization methods appealing, however, is that they allow one to take biomechanical and physical principles and directly employ them in generating motions. This focuses the motion creation process on what end up often being relatively conceptually coherent reasons for why an animal moves in the way that it does, leaving what end up often being relatively complicated facts about the specifics of the motion itself up to the computer.

The optimization approach to the problem of motion generation leads naturally to a particular mathematical framework upon which the methods in this thesis are all based. I'll give it here only in a very general form, leaving a more detailed consideration of the numerous details and variations for later. If the symbol M is used to refer to a motion, then the overall mathematical formulation of an optimization approach to motion generation can be written:

$$\begin{aligned} M^* = \underset{M}{\text{argmin}} \quad & \text{effort}(M) \\ \text{s.t.} \quad & \text{physics}(M) \end{aligned} \tag{2.1}$$

Essentially this defines the *optimal motion* M^* as the motion which obeys the laws of physics, and which subject to that restriction takes as little effort as possible to perform.

Many of the terms in equation 2.1 have been left intentionally vague. In particular I've omitted details on how the motion M is represented, how the constraint that M^* obey the laws of physics is enforced, and what specifically is meant by the rather ambiguous term "effort". This vagueness reflects the fact that all of these terms can potentially be defined in different ways, leading to a variety of different algorithms for solving the resulting optimization problem.

Roughly speaking, there are two broad classes of algorithms which aim to either solve or approximate a motion M^* as specified by equation 2.1 (figure 2.2). The first of these classes I will refer under the umbrella term of *dynamic controller* algorithms, and synthesize a stream of motion while continuously adapting to the character's current state. This might

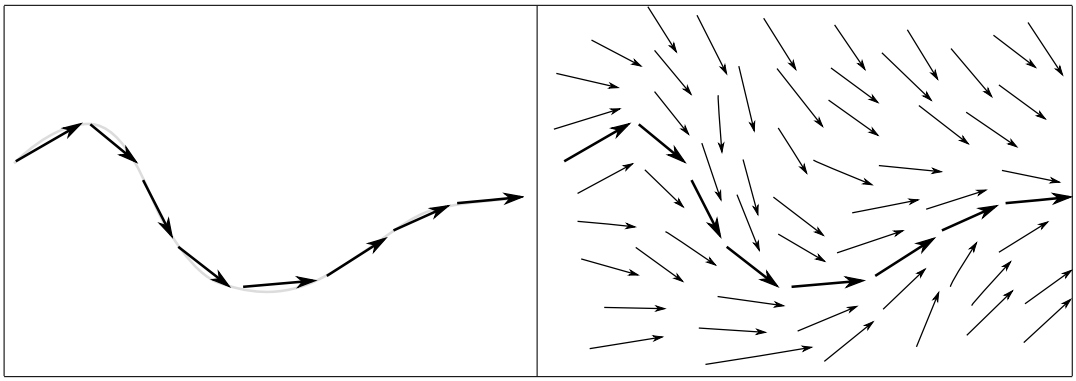


Figure 2.3: A conceptual illustration of the difference between trajectory synthesis and controller generation approaches to motion synthesis. In each case the animal is illustrated as having two DOFs, and thus a motion can be represented as a curve in 2D space (in reality, the simplest animal I test with has eight DOFs, but pretending an animal's pose can be represented by just two DOFs makes the figure much easier to draw!). On the left a trajectory synthesis method is illustrated, the output of which is simply a single motion, represented here by a curve through the space of the animal's DOFs. A controller generation approach, however, behaves more as illustrated on the right and provides an entire vector field across the state space, and a motion can be synthesized by starting at any state and following the 'flow' of this vector field.

be viewed as roughly analogous to how an actual creature moves by choosing its motion at each moment in time in order to adaptively keep its balance. The other major class of algorithms I will refer to as *trajectory synthesis* algorithms, and generate an entire motion all at once. This is somewhat similar to how an artist might create an animation by creating and refining the entire motion until a satisfactory result is achieved. In this thesis I have chosen to use this latter approach to motion synthesis, but to better understand this choice it's useful to briefly review the dynamic controller approach.

2.2.1 Dynamic Controller Approaches

Although there are a wide range of different specific algorithms which fit under the umbrella of dynamic controller approaches, almost all of them share some important features in common:

- The motion M^* is solved for one frame at a time.

- physics(M) is enforced by *simulation*.

- The motion is controlled by torques at the character's joints.

The reason for and function of each of these three properties can be made more clear with a high-level description of how a motion would be synthesized by a dynamic controller algorithm.

In a dynamic controller algorithm, the character starts out in some state, which in this case can be thought of as fully defined by the combination of a pose and a velocity. That is, a character starts out with some specific vector of values defining the position of each of its degrees of freedom, as well as the rate of change for each of these degrees of freedom. Starting from this initial state, and given values for the torques applied at each of the character's joints, a physics simulator can be used to solve for the character's new state some (generally small) amount of time into the future. Since this simply brings the character to a new state, the same procedure can be applied again, iteratively generating one frame of animation at a time. Because a physics simulator is used to bring the character from one state to another, the resulting animation automatically obeys the laws of physics.

The job of the control algorithm, and indeed the primary technical difficulty with this approach, is how to choose the particular torques to apply to the character's joints in order to achieve a desired motion. The primary reason why this poses a difficult problem is because legged locomotion is *underactuated*, meaning that there are more ways in which a character can move than there are in which it can apply torques to control this motion. In particular, an animal has no direct muscular control over its center of mass, but must instead must use the forces arising from contacts between its feet and the ground to indirectly control the motion of its center of mass. One manifestation of this underactuation is the possibility that an animal may fall over if it doesn't perform a motion correctly.

Creating a motion controller which can locomote while keeping the character form falling over is a central problem in designing locomotion controllers. Over the years many methods have been developed to make controllers of this sort easier to create, including techniques in which controllers can be reasonably authored by hand [63, 91, 12, 14, 38, 15] and approaches for automatically creating and tuning controllers [89, 56, 79, 80, 81, 17]. Still, as it stands

there are no available approaches which can automatically create controllers for a wide range of different animals without recourse to motion capture data.

2.2.2 Trajectory Synthesis Approaches

The primary alternative to dynamic controllers for motion synthesis is trajectory synthesis. This too is an umbrella term covering many algorithms which are rather different in their details, but there are nevertheless some common aspects to these approaches:

- All the frames in M^* are solved for simultaneously.
- physics(M) is enforced by *constraint satisfaction*.
- The motion is controlled by angles at the character's joints.

In truth, it is only the first of these properties which characterize an algorithm as a trajectory synthesis algorithm, but the latter two properties are still shared by most (but not quite all) trajectory synthesis algorithms.

Trajectory synthesis algorithms generally follow the pattern of a classical optimization algorithm. As before it's useful to see how such an algorithm proceeds at a high level. The first step in synthesizing a motion using a trajectory synthesis technique is to define the basic structure of the desired motion. Doing so requires the user to pre-specify a few basic aspects of the animation. The most obvious of these aspects is the type of character (or characters) performing the motion and some mathematical description of the goal which the motion is trying to achieve.

In practice there are a few other aspects of the animation which also need to be specified so that traditional optimization methods can be applied to the problem. In particular many of the most efficient optimization methods are restricted to searching over some n-dimensional Cartesian space. Since each motion M can represented by a vector of real numbers, this in principle allows these optimization techniques to be used to search over the space of motions. The catch, however, is that the dimension of this vector cannot change during the optimization process, requiring any aspects of the animation which might require

a change in the number of values needed to describe a motion to be fixed in advance. The most common of these involve fixing the number of frames in the animation and often additionally specifying which of the character's feet are in contact with the ground (or other elements in the environment) at each frame.

With these few aspects to the desired motion fixed in advance, the space of possible motions forms an n-dimensional Cartesian space, where 'n' is the number of variables needed to describe a motion – typically on the order of a hundred to a few thousand. Although in principle a motion might be described in a multitude of different ways, for instance by joint torques at each frame and a specific starting state, in practice most methods use something resembling the method illustrated in figure 2.2 where the motion is parameterized by the rotations at each of the character's joints at each frame. In addition many trajectory synthesis approaches require some additional variables in order calculate properties of the motion related to physics, but these are generally of a more algorithm-specific nature.

The real meat of a trajectory synthesis algorithm lies in how to search the space of all possible motions for the one particular motion which will (hopefully) best achieve the user's desired result. Fortunately this problem fits well within the domain of classical optimization theory, which contains a is relatively well-developed theory for solving problems of this sort and an extensive array of tools available for doing so.

The details of a particular optimization technique can be both varied and involved, but for the most part an optimization will start with an initial guess for the motion, and then gradually refine this guess so that the resulting motion better satisfies some desired property. Typically this 'desired property' is representing by specifying a *cost function* (also called an *objective function*) which for each possible motion yields a single real number representing how well the motion satisfies the desired properties, with lower values of the cost function indicating that the properties are better satisfied than for higher values. An optimizer thus gradually refines the initial motion so that each successive version gives a lower value for the cost function, while taking care to also better satisfy the constraints representing the physical validity of the motion.

Trajectory synthesis methods have several advantages over dynamic controller approaches when it comes to generating a motion for an input animal. Perhaps the most important

of these is that only a single motion needs to be generated, whereas a controller must be able to cover a larger space of possible states since no fixed 'starting state' is assumed for the animal. This eases the computational and algorithmic burden which must be carried by trajectory synthesis methods. Secondly, in trajectory synthesis methods the motion does not necessarily need to satisfy the laws of physics until a final result is obtained, but the initial and intermediate motions generated by the optimizer may violate the laws of physics if convenient. In principle, and often in practice, this allows for the final motion to be found more efficiently and for this motion to better minimize the specified cost function.

In the next chapter of this thesis I will develop in more detail a formulation for a trajectory synthesis method which is the first capable of automatically synthesizing physically realistic motions for a wide range of different animals. The following two chapters will elaborate on this trajectory synthesis method by developing techniques to improve the realism of the generated motions and to automatically generate a family of interconnected motions which can be used to interactively control an animal in real time.

Chapter 3

ANIMAL MOTION SYNTHESIS

The most fundamental concern in the synthesis of animal locomotion is, rather self evidently, the question of how such motions might be synthesized at all. It is obvious that such motions are at least theoretically possible to generate, since real-world animals (including ourselves) routinely do so with a remarkable grace. As is unfortunately the case with many activities which seem to come to us with a natural unconscious ease, the commonplace nature of locomotion belies the substantial difficulties in generating these motions computationally. As the ability to synthesize locomotion for an arbitrary animal animal forms the backbone of the more involved techniques presented later in this thesis, and indeed a number of hypothetical future directions that work on animal motions could take, it is worth considering the properties that such a synthesis method should have.

In essence, this question about the desirable properties in a locomotion synthesis method is just a question of how the shape of an animal is related to the synthesized motion. It is of course clear that any locomotion synthesis method should somehow capture the interrelationship between the form and the motion of an animal. What is perhaps less clear at first glance are the various shades of refinement in how intricately this intertwining between an animal's form and motion is modeled.

Even modeled at its most coarse level, it's clear that some manner of interrelating the form of an animal to its motion is necessary. After all, if an animal has four legs they're going to all be involved in the motion somehow, no matter how crude the animation method. In addition, we would expect that anything passing as an animation method would also satisfy some other properties, such as having an animal's feet at least approximately come into steady contact with the ground in a periodic manner as the animal moves forward, or ensuring that a leg lifts above the ground during its swing phase. All of these are, strictly speaking, manners in which an animal's form leads to its motion.

The common property behind these coarse properties relating an animal's shape to how it moves is that they are *kinematic*. The term "kinematic" in this context simply refers to the fact that these aspects of an animal's motion are only dependent on functions of the position of an animal's limbs during a motion, and do not depend at all on the laws of physics or related principles. In some sense, a purely kinematic view of motion attempts to model how a motion looks without addressing why it looks that way. Although such kinematic approaches most assuredly have their place in animation, there is good reason to doubt their effectiveness when applied to animal locomotion.

Although the kinematic properties of a motion are certainly an important factor in animal locomotion, focusing on them exclusively leaves out the *dynamical* aspects of a motion – that is the aspects of a motion relating to the laws of physics. It is these dynamical properties of motion which necessitate the coordination of leg movements to keep balance, the spring and release of leg posture during a running gait, regulating center of mass position to be over the feet an average, and many of aspects critical to the natural appearance of motion. Although, at least in theory, one might hope to capture these nuances using purely kinematic means, it is not clear how this might be done and at best it seems a daunting task.

Including information about the laws of physics into the motion synthesis process, however, provides a succinct and powerful means to incorporating realism into the motion synthesis process. Furthermore, if one wishes to synthesize motions for any sort of application related to biology or biomechanics, the impact of physics on the motion of real animals is so fundamental that is it almost a necessity that the synthesizes motions be informed by the laws of physics.

All of the motion synthesis methods presented in this thesis will therefore be focused on synthesizing motions which are completely consistent with the laws of physics. More specifically, an ideal synthesis method, and indeed all of those presented in this thesis, will share the following properties:

1. Can be applied to a wide range of different animals.

2. Synthesis is fully automatic.

3. The synthesized motions are consistent with the laws of physics

4. The synthesized motions are biomechanically motivated.

Synthesizing physically valid animal locomotion is in general substantially harder than employing a purely kinematic synthesis technique. This is because although there are many ways in which an animal might try to move, only a small fraction of these result in successful locomotion. Depending on the representation used, the vast majority of all possible motions will instead either disobey the laws of physics, or cause the animal to promptly fall over.

In the remainder of this chapter I will introduce an approach to animal gait synthesis which is the first to reliably and automatically create physically valid motions for a wide range of different animals. This method falls under the umbrella of trajectory synthesis approaches (section 2.2.2) and thus synthesizes the entire motion 'all at once' using an iterative optimization approach.

After a bit of background on previous approaches to the problem of gait synthesis, I will introduce the my optimization approach in two parts. First considering the (mathematically simpler) problem where the timing of the contacts between the animal's feet and the ground is fixed. Next, I will introduce another optimization method which can solve for the optimal timing with which the animals feet should contact the ground. Finally, I'll conclude with an additional extension which allows not only the solution of motions, but also for the shape of the animal itself to be refined in order to better achieve some user-specified goal.

These approaches are able to synthesize motions for a wide range of different animals, including both those resembling real-world bipeds and quadrupeds as well as completely fictional creatures with, for example, just a single leg, or with a set of five legs. The motions and shapes generated by my approach appear visually plausible and do not require any per-animal user authoring or preset motion patterns. As far as I am aware, this is the first method to achieve this fully automatically for non-simplified characters with highly underactuated motions. Although the biomechanical model employed here somewhat approximate, this method's ability to perform a *de novo* synthesis from basic physical principles is a first step toward frameworks (some of which will be considered later in this thesis) in which more

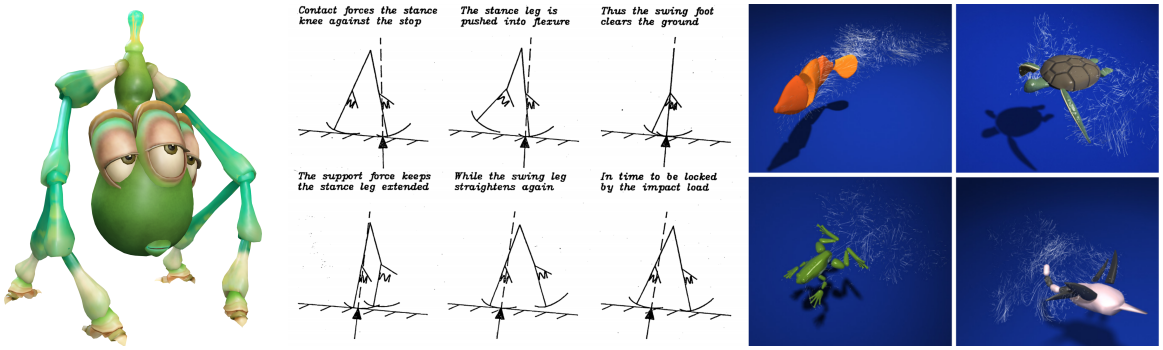


Figure 3.1: Examples of three different approaches for synthesizing motion without data. From left to right: Generalization on artist-designed motions [27], motion synthesis for a simple creature [53], and synthesis of well-actuated motions [73].

accurate biomechanical models may be applied to create truly accurate motions for animals alive, extinct, and imaginary.

3.1 Background

The difficulties posed by underactuated motions have typically limited existing locomotion synthesis approaches which do not rely on captured data to one of three different categories (figure 3.1):

1. Manually crafted motions.

2. Motions for simple creatures.

3. Well-actuated motions/creatures.

For instance, the motions used in the video game SPORE were generalized from manually designed motions [27], and some early physical simulations of locomoting creatures were manually crafted as well, of which [63] is a good example. More recently, SIMBICON has provided a framework in which successful locomotion patterns have been crafted for both bipeds [91, 12] and quadrupeds [14]. Other recent techniques have focused on providing simpler and easier ways to manually design gaits and other locomotion patterns [38, 15].

Examples of existing research which do not rely on hand-authoring of the animal's motion include some cases where the animal in question is modeled as a highly simplified abstraction, such as a center of mass plus massless non-jointed legs or a creature with some other very simple structure [53, 4, 30, 71, 61, 76]. Other examples of successful from-scratch synthesis of animal motion include cases where the motion is well-actuated. This includes non-terrestrial locomotion such as swimming [20, 73] or flying [88], or 'wriggling'/'crawling' type terrestrial locomotions [68]. Although some examples of automatically synthesized motions for complex characters do exist [89, 56, 79, 80, 17], they have thus far only been applied to bipeds, and no pre-existing techniques can reliably be applied to a wide range of different animals.

As mentioned in section 2.2 synthesis methods based on optimality are well suited to automatically generating accurate gaits for an animal without relying on any pre-captured data. The primary intuition behind this is that performing low-effort locomotion well is critical to survival for a great many terrestrial animals. For animals in this class (for instance, animals which undergo long migrations), evolutionary forces have presumably shaped their motion to be at least near-optimal for locomotion. Furthermore, even animals which do not necessarily depend on highly efficient locomotion as a key to their survival are unlikely to waste exorbitant amounts of energy in their basic gaits.

This implies that synthesis methods based on optimality stand a chance of capturing the deeper biological reasons behind why an animal moves in the way that it does. This alleviates the need for any manual tuning of the motion, and allows the salient traits in a motion to be easily applied to animals of diverse shapes. For instance, while optimality-based criteria such as the emphasis placed on minimizing muscle strain in a motion can be applied equally to both bipeds and quadrupeds, this generalization is substantially trickier for many non-optimality-based terms as such the pattern of torques applied at each joint. Furthermore, within an optimality-based approach it is conceptually simple to generate a range of different motions (such as walks, runs, and turns) from the same cost function, while such a generalization is often trickier in other approaches.

The underactuation of most terrestrial animals combined with the relative complexity of these animals poses serious difficulties in accurately solving for an optimal motion. For

highly simplified abstractions of animals, this optimization process can sometimes be solved by a brute force or other simple search over the parameters of the motion [30, 53, 71]. However these techniques do not generally scale to more complicated animals, and more sophisticated optimization methods are needed.

The type of optimization techniques which might be brought to bear on the problem of animal locomotion synthesis depends a great deal on the way that a motion is represented. As outlined in 2.2, there are two classes of motion representations commonly employed today, each with its associated methods used to solve the resulting optimization problem. The class of approaches known as controller methods treat motion as something which is continually generated 'on the fly' in response to potentially changing conditions, information, and state. The optimization problems associated with controller-based approaches to motion synthesis are often (although not always) solved with 'direct-search' optimization techniques which do not rely on the computation of any useful derivatives. The second class of approaches are trajectory synthesis approaches and treat motion as something which can be planned 'all in one go' in light of pre-existing information about the environment in which the motion occurs. The associated optimization problems for this class are often (although not always) solved with optimization techniques such as the method of spacetime constraints [84] which require the existence of useful derivatives. In this and the following section I will be focusing on a the trajectory representation of motions, considering the controller representation in chapter 5. For the moment, however, I will consider some of the existing approaches used to solve for these motions.

Although direct search techniques have been used successfully in many circumstances, solving for motions of simplified creatures [4], adapting human motions [28], or animating creatures which move with well-actuated motions [73, 88, 68], they are trickier to apply to generating locomotion for real creatures without requiring motion data or manual tuning. The most successful methods so far in this regard can generate motions from scratch, but are limited to the type of animal that they can be applied on (generally bipeds) [79, 80]. Furthermore existing methods in this class do not generally allow for easy control over the nuances of the motion, but instead only consider high-level features such as the position of foot contacts or the height of the character's center of mass, mapping these high-level to a

complete motion with a control mechanism.

The other dominant paradigm for synthesizing motions is the optimization-based approach of spacetime constraints [84]. This approach finds a motion for a character by solving a large constrained optimization problem. In this formulation the cost function is generally loosely related to minimizing the energy required to perform a motion, subject to a set of constraints enforcing that the resulting motion satisfy the laws of physics. Unlike direct-search approaches, most approaches using this technique require the computation of useful derivatives for the cost and constraint functions. This limits the scope of the problems that can be solved using spacetime constraints, but potentially allows for more nuanced and complicated motions to be solved for efficiently.

Because of the ability of spacetime constraints to solve for a full motion directly (without need for an intermediate control mechanism) and to manage complex and poorly-articulated motions, my proposed methods for solving for animal motions are based on this technique. There are, however, some significant hurdles which need to be overcome. In particular existing work using spacetime constraints has typically been limited to simplified creatures or has required motion capture in order to provide a 'good starting point' for the optimization process – thus being employed to modify an existing motion rather than synthesize a new motion in the absence of any data [62, 17, 48, 49, 64, 84]. Although a few approaches have been suggested to ameliorate this requirement for an initialization motion by using a database of motion capture to define the space of reasonable motions [65, 82] or with a specialized dynamics formulation [17], no pre-existing methods can solve for a motion for a new animal for which there is no data. Furthermore preexisting approaches also require the order and approximate duration of the foot contacts to be specified in advance (although some recent research published a few years after the work presented in this chapter has also allowed the contact order and timing to be optimized over [58]).

In the remainder of this chapter I will begin by describing the basic spacetime constraints formulation I have used for animal gait synthesis (section 3.2). This optimization can successfully solve for an animal's gait from scratch, but requires information to be prespecified about the timing of the contacts between the animal's feet and the ground. To overcome this limitation in section 3.3 I will describe an optimization technique which can

optimize over the timings of the contacts between the animal's feet and the ground, as well as increase the robustness of the optimization to a poor initialization. Finally, in section 3.4 I will provide an extension to the spacetime constraints optimization which allows both the shape of the animal and its gait to be optimized over simultaneously.

3.2 Continuous Optimization

At a high level, the optimization method presented here takes as input a description of the shape of an animal along with a few simple user-supplied goals and outputs a physically realistic gait for the input animal. The optimization process itself consists of two components. The first of these is a derivative-based nonlinear constrained optimization problem particularly well suited to solving for an animal's gait – providing that the timings of the foot contacts involved in the gait are fixed in advance. To optimize over foot timings (and to achieve greater robustness) this optimization method is then integrated into a secondary outer-loop optimization. This 'outer loop' of the optimization process will be described in section 3.3, and the more fundamental core optimization which assumes that foot contact times are fixed will be described first.

The core to my optimization is formed by a derivative-based spacetime constraints approach. While the formulation given here is not fundamentally different from previous approaches, it is notable in two key respects. Firstly, the optimization has been specifically focused on generating a closed-loop gait cycle, and it is capable of solving for such a motion without a good initialization. Secondly, in order to avoid unrealistic results without relying on a good initialization I have been more thorough in the optimization's formulation than is typical.

The input to the optimization is defined as a kinematic tree of connected limbs (as illustrated previously in figure 2.1). Each limb is modeled as a cylinder with a length, radius, and mass. Pairs of limbs are connected by *joints* which define a parametrized transformation from the endpoint of the parent limb to the endpoint of the child limb. Generally this transformation consists of a rotation followed by a translation along the length of the child limb. I further define a special joint at the root giving the global rotation and translation of the animal. Some care can be required in defining the correct mathematical

transformations for the different types of joints, and a full list of these transformations can be found in appendix A. In addition the the user must also specify the limb endpoints corresponding to the animal's feet and the limb corresponding to its head (if any).

Given such a description of an animal's motion over a single gait cycle can be defined with a set of variables describing the parameters for its joints at each of a fixed number of frames in a manner similar to that illustrated in figure 2.2. For each foot, at each frame when it is in contact with the ground, six additional variables are defined giving the force and torque exerted on the foot via this contact, denoted by $\mathbf{f}_{i,j_c}$ and $\mathbf{t}_{i,j_c}$. For each rotational joint parameter three further variables are included defining its passive actuation characteristics, represented by a spring stiffness, rest state, and a dampening constant. Additionally in sections 3.3 and 3.4 some additional variables will be introduced to control when each foot is in contact with the ground and to allow deformations in the shape of the animal itself.

Provided this representation of a motion, recall the high-level sketch of a motion optimization problem given in equation 2.1:

$$\begin{aligned} M^* = \underset{M}{\text{argmin}} \quad & \text{effort}(M) \\ \text{s.t.} \quad & \text{physics}(M) \end{aligned}$$

This particular equation was structured to be general enough to cover a very wide range of optimization-based motion synthesis methods for physically valid character motion. This generality, however, necessitated a relatively vague equation. The specific form of this equation used in this thesis to synthesize motions is a relatively direct translation of the above equation into a *nonlinear programming problem*, yielding an optimization problem with a form which has come to be termed a *spacetime constraints* problem:

$$\begin{aligned} M^* = \underset{M}{\text{argmin}} \; & \frac{1}{n} \sum_i cost(M(f_i)) \\ \text{s.t. } & g_j(M(f_i)) = 0 && \forall j, i \\ & h_k(M(f_i)) \leq 0 && \forall k, i \end{aligned} \tag{3.1}$$

Typically $cost(M(f_i))$ is taken to be the sum of the joint torques or muscle forces exerted by the character at frame f_i and the constraint functions g_j and h_k enforce that the resulting motion satisfy the laws of physics. This optimization problem falls within the class of nonlinearly constrained nonlinear programming problems. Although there are well-studied techniques for solving problems in this class, the ones arising in spacetime constraints are replete with practical difficulties such as local minima and poor conditioning. A successful concrete implementation of equation 3.1 thus requires a great deal of care, particularly when the same implementation must be applicable to a wide range of different animals.

Although the above equation neatly describes the general form of an optimization problem solving for a gait, there are numerous details regarding its parameterization and the specifics of the functions involved that must be addressed in any successful implementation. In brief, these issues consist of enforcing the resulting motion to be cyclic, defining the constraint functions g_j and h_k, and determining the cost function *cost*.

Enforcing cyclicity is probably the simplest of these concerns. To easily optimize for a cyclic gait the most convenient method is take a "treadmill" approach. That is, to derive a gait for an animal running at some velocity $\mathbf{V}_{gait}$, an optimization is solved where the animal stays stationary overall and instead has its feet moving at $-\mathbf{V}_{gait}$ when in contact with the ground. This is dynamically equivalent to an animal running on stationary ground and allows the easy definition of an optimization for a cyclic motion by applying all terms which rely on temporal derivatives, such as the cost function and the physical constraints (which both rely on accelerations to calculate the internal forces with inverse dynamics), in a cyclically looping manner.

Shorty a precise mathematical definition of *cost*, as well as to define the constraint functions g_j and h_k enforcing the laws of physics will be provided. To aid in writing these later equations, I will define here some common terms which will appear in the equations:

m the default mass of the animal

q a single joint degree of freedom

f,t force and torque, respectively

$p(i,j)$ position of bone endpoint (node) or joint j at frame i

$v(i,j)$ the linear velocity of node j at frame i

$R(i,j)$ The 3×3 rotation matrix of node j at frame i

In addition, unless noted otherwise, I will use i to index over frames, j to index over joints or bone endpoints, l to index over limbs, and j_c to index over ground contacts. So for instance $\mathbf{f}_{i,j}$ will refer to the force at joint j in frame i.

This optimization utilizes several dynamic and kinematic terms, discussed next. For the moment, note that the current formulation requires that the frames at which each foot is in contact with the ground be fixed in advance. In section 3.3 a method will be introduced by which the foot contact times can be allowed to vary in a global optimization outer loop.

3.2.1 Constraints

My motion synthesis method formulates the gait synthesis problem as a constrained optimization, and as such a method is required to mathematically specify the constraints. These constraints can be of two forms: *equality constraints* and *inequality constraints*. Intuitively, equality constraints enforce that some property of the motion hold exactly. For example, the constraint enforcing that when a foot is supposed to be in contact with the ground then it should be exactly on the ground is an equality constraint. Inequality constraints, on the other hand, denote certain motions as invalid by checking if some property of the motion exceeds some given bounds. For instance, the constraint stating that no part of the animal can pass below the ground is an inequality constraint.

The most common mathematical method of specifying the constraints in an optimization problem, and that used in this thesis, relies on *constraint functions*. For equality constraints, a constraint function g takes an input a single frame $M(f_i)$ within a motion and returns a real number such that the motion is considered valid only if $g(M(f_i)) = 0$. An inequality constraint h has a similar form, with the difference that the motion will be considered valid only if $h(M(f_i)) \leq 0$. If the motion is valid as far as a particular constraint function is concerned, then the constraint is said to be *satisfied*. For a motion to be considered valid,

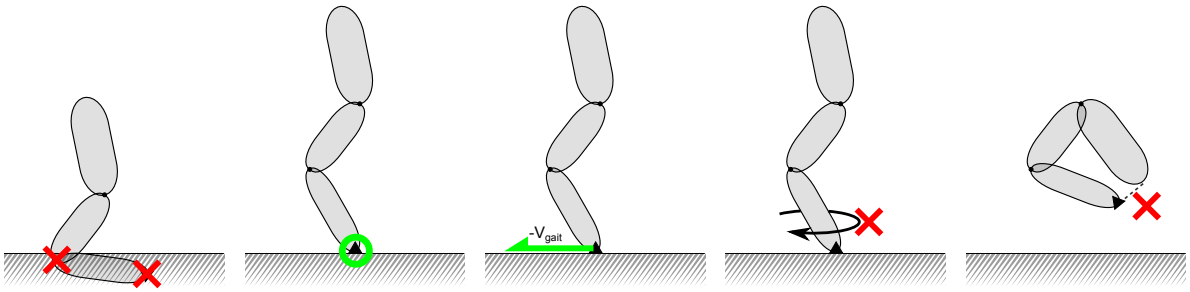


Figure 3.2: Graphical illustrations of the kinematic constraints used in a spacetime constraints optimizations. From left to right: animal can't penetrate ground, foot must be level with ground when in contact, foot must have correct velocity when in contact, foot can't twist when in contact with ground, and non-intersection of an animal's limbs implemented as a minimum distance between specified pairs of limb-endpoints.

all of the constraint functions must be satisfied. Additionally, the most efficient constraint optimization methods require that the constraint function be differentiable, so all the the functions given in this section admit the efficient computation of derivatives.

Kinematic Constraints

The simplest and most conceptually straightforward of the constraints used in the optimization (equation 3.1) are those enforcing a kinematic validity to the motion. Each of these constraints is used to prohibit some class of kinematically inadmissible motions, and in total five types of constraints are required to ensure the kinematic validity of the motion (figure 3.2):

1. The animal must stay entirely above the ground

2. Any foot which is in contact with the ground must be physically positioned level with the ground plane.

3. When a foot is in contact with the ground its velocity must be $-\mathbf{V}_{gait}$

4. When a foot is in contact with the ground it should not twist with respect to the ground.

5. The animal's limbs cannot pass through each other.

Assuming the ground is flat and that the thickness of each of an animal's limbs is ignored, instead approximating them with a long but thin rod, then the first constraint that the animal never pass below the ground can be simply approximated. This is achieved by enforcing the constraint hold independently for each node (bone endpoint) on the animal and for each frame in the animation:

$$\forall_{i,p} : p(i,p)_y \geq 0 \tag{3.2}$$

In addition to the constraint that the animal not pass through the ground plane, there are additional constraints which apply to an animal's foot when it is contact with the ground. Unlike the ground-penetration constraints, these constraints only apply to a foot when it is in contact with the ground. The simplest two of these contact constraints enforce that when a foot is supposed to be in contact with the ground that the position of foot should actually be *on* the ground:

$$\forall_{i,j_c} : p(i,j_c)_y = 0 \tag{3.3}$$

and that when in contact the foot should be at rest relative to the ground. Since the optimization takes a 'treadmill' approach to easily enforce cyclicity, this implies that the foot's velocity when in contact must be $-\mathbf{V}_{gait}$:

$$\forall_{i,j_c} : v(i,j_c) = -\mathbf{V}_{gait} \tag{3.4}$$

There is one additional constraint pertaining to the animal's feet when in contact with the ground that relates to geometrically modeling each foot contact as occurring at a single point. This simple approach does not allow more detailed modeling of a foot's structure such as the heel-toe roll in human gaits or the use of the toes to push off the ground in some animal gaits. Fortunately most non-human animals have small ground contact areas, but it

is still useful to explicitly define a preferred orientation for a foot on the ground by adding constraints enforcing each foot's rotation about the y-axis to be zero when in contact:

$$\forall_{i,j_c} : R(i,j_c)_{x,z} - R(i,j_c)_{z,x} = 0 \tag{3.5}$$

For efficiency and ease of differentiation the constraint enforcing the non-self-intersection of an animal's limbs can be enforced approximately by having the user provide a list of pairs of points in the animal $(p_{1,1}, p_{1,2}), \ldots (p_{n,1}, p_{1,2})$ and enforcing a minimum distance constraint between these points:

$$\forall_{i,k} : \|p(i,p_{k,2}) - p(i,p_{k,i})\|^2 \geq r^2 \tag{3.6}$$

In practice it is generally sufficient to define one pair of points corresponding to the position of each of the pairs of an animal's feet, and another analogous set of pairings for an animal's knees.

Finally, to remove the translational invariance of the optimization and keep the resulting gait from drifting arbitrarily it is computationally handy to add a term to keep the average position of the animal along the x-z plane centered at the origin.

$$\sum_i p(i,root)_x = \sum_i p(i,root)_z = 0 \tag{3.7}$$

Strictly speaking this final constraint is not necessary, and a motion would be kinematically valid even if it were not satisfied. Nevertheless it is useful in a computational sense to remove unnecessary degrees from the optimization, and the optimization tends to converge more quickly and robustly with this constraint than without it.

Mass Normalization

In the following description of the dynamical constraints and objective function used in the optimization, some terms are normalized by the default mass of the animal. This helps keep the optimization better conditioned for animals of large mass and is similar to methods employed by [71, 28]. This normalization is achieved by treating the optimization

variables for the ground reaction forces as scaled by the mass of the animal $\frac{\mathbf{f}_{i,j_C}}{m}$ and $\frac{\mathbf{t}_{i,j_C}}{m}$ and introducing a similar multiplicative normalization term to the other equations which deal with dynamics. This allows all intermediate computations to be performed in standard SI units – a benefit in defining terms which depend on constants such as the maximum stress a muscle can support, as discussed later in section 3.4.

Dynamic Constraints

In addition to the previously described kinematic constraints, my optimization formulation makes use of a number of *dynamic constraints*. These constraints are related to the laws of physics, and enforce that the motion obey properties such as Newton's laws or Coulomb's law of static friction. The dynamic constraints are slightly more involved to specify then the kinematic constraints, and much more difficult to computationally enforce in the optimization, but their inclusion is vital to the generation of accurate motions.

I phrase the dynamical constraints of my gait synthesis method with a Newton-Euler formulation reminiscent of Fang and Pollard [17] by expressing the dynamical constraints about the animal's center of mass, yielding a total of six dynamical constraints per frame. In order to calculate these constraints more efficiently each limb is approximated by a point mass located at the limb's center of mass. Although potentially significant in a theoretical sense, in practice the visual effects of this approximation are almost never noticeable for real-world animals.

Within this formulation there are four different types of constraints which must be enforced to ensure that a synthesized motion is consistent with the laws of physics:

1. The time-derivative of the animal's linear momentum must match the sum of the external forces acting upon the animal.

2. The time-derivative of the animal's angular momentum must match the sum of the external torques acting upon the animal.

3. The contact forces between the animal's feet and the ground should obey Coulomb's friction model.

4. The contact torques between an animal's feet and the ground should be small enough to be explained by the size of the foot.

The first two of these constraints directly reflect Newton's second law of motion. At each frame, the time derivatives of the animal's linear and angular momenta must be equal to the net force and torque on the animal respectively:

$$\forall_i : \dot{\mathbf{p}}_i = mg + \sum_{j_c} \mathbf{f}_{i,j_c} \tag{3.8}$$

$$\forall_i : \dot{\mathbf{L}}_i = \sum_{j_c} \left(p(i,j_c) - \mathbf{CM}_i\right) \times \mathbf{f}_{i,j_c} + \mathbf{t}_{i,j_c} \tag{3.9}$$

Where p_{cm} and v_{cm} give the position and velocity of a limb's center of mass and CM_i gives the position of the center of mass of the animal at frame i.

The only external forces acting on an animal during its gait cycle are those due to gravity and the forces arising from the contact of the animals feet with the ground. These contact forces cannot take arbitrary values, but must be constrained to be admissible under Coulomb's friction model:

$$\forall_{i,j_c} : \frac{1}{m}\left(\mu\, \mathbf{f}_{i,j_c\perp} - \left\|\mathbf{f}_{i,j_c\parallel}\right\|\right) \geq 0 \tag{3.10}$$

Where $\mathbf{f}_{\parallel}$ and $\mathbf{f}_{\perp}$ represent the components of the contact force parallel and perpendicular to the ground respectively.

Similarly, the torques resulting from a ground contact cannot be arbitrary, but must be small enough to be explained by the size of the foot. This requirement can be simply approximated by constraining the torque components of the ground reactions to lie within an ellipsoid scaled by the force component of the reaction, approximating the behavior of a foot which has an area of contact with the ground, even though the foot in the optimization is still geometrically treated as a point:

$$\forall_{i,j_c} : \frac{1}{m}^2 \left(\mathbf{f}_{i,j_c}{}^2_{\perp} - \left\|\frac{\mathbf{t}_{i,j_c\parallel}}{\nu_b}\right\|^2 - \left(\frac{\mathbf{t}_{i,j_c\perp}}{\nu_t}\right)^2\right) \geq 0 \tag{3.11}$$

Where μ is the coefficient of static friction and ν_b and ν_t give bounds on the maximum allowed torques in the directions parallel and perpendicular to the ground respectively. Reasonable values of these constants are $\mu = 1.0$ and $\nu_b = \nu_t = 0.02$ and can be held fixed for all of the animals synthesized.

3.2.2 Passive elements

In addition to being able to actuate their joints through muscular exertion, many animals also have musculo-skeletal structures that behave like springs or dampers and passively actuate these joints. The importance of these in reproducing styles of walk in humans was noted in [48], which may also be referred to for a more in-depth discussion of these terms.

Passive elements can be integrated into the optimization by representing the total torque at each joint as the sum of a passive and an active component:

$$\mathbf{t} = \mathbf{t_a} + \mathbf{t_p} \tag{3.12}$$

Where $\mathbf{t_a}$ is the component of the torque arising directly from muscular exertion while $\mathbf{t_p}$ is the component arising from passive actuation. This allows the animal to achieve more efficient gaits and is reflected in the objective function to be introduced shortly in section 3.2.3.

$\mathbf{t_p}$ is computed for a gait for a gait by including three variables for each joint degree-of-freedom, $\mathbf{q}_j$, in the animal: A spring constant k_{sj}, a spring rest length, $\bar{\mathbf{q}}_j$, and a dampening coefficient k_{dj}. The passive force for that degree of freedom can then be written as:

$$\mathbf{t_p}_j = -k_{sj}(\mathbf{q}_j - \bar{\mathbf{q}}_j) - k_d\dot{\mathbf{q}}_j \tag{3.13}$$

Although for the most part the inclusion of passive elements does not drastically alter the overall form of a gait, it does make the resulting motions significantly smoother and more believable. These terms will later play a more important role in improving the realism of gaits in chapter 4.

3.2.3 Objective

So far I have described the constraints required to satisfactorily distinguish valid motions from invalid motions. There are, however, still many different valid motions possible under these constraints. Some of these valid motions correspond to more realistic gaits than others, so some method is still required to rank the desirability of the motions within the space of all valid motions. This is achieved with a *cost function* (or equivalently an *objective function*) which for each frame in a motion $M(f_i)$ returns a real number $cost(M(f_i))$ such that lower values of this cost correspond to more desirable motions. The optimization then searches for the single motion which minimizes the sum of this cost for each of its frames while also lying within the space of valid motions. Since a cost function in some sense ranks how 'realistic' a motion is, its definition is crucial to successful gait synthesis.

The objective function used in the basic form of the optimization, and which is sufficient to generate qualitatively plausible gaits, can be written as the sum of four terms, each of which expresses a certain preference for some motions over others:

1. The animal should minimize the sum of the torques at each joint.

2. The animal should minimize the velocity of its rotation at each joint.

3. The animal should keep the position of its head relatively still.

4. The animal should avoid excessive rotations of its head.

The first of these terms is standard in spacetime optimizations and (approximately) minimizes the muscular exertion of the animal. This muscular exertion is represented by the active torque required at each joint to perform a given gait. The torques used are found analytically from the motion and contact forces using inverse dynamics based on the recursive Newton-Euler formulation [18]:

$$C_1 \frac{1}{m}^2 \sum_{i,j} \left\| \mathbf{t}_{\mathbf{a}i,j} \right\|^2 \tag{3.14}$$

The second term penalizes high-velocity joint motions and is necessary to avoid low-torque but unrealistic 'wiggling' motions that would otherwise occur in the flight phase of some gaits:

$$C_2 \sum_{i,\mathbf{q}} \dot{\mathbf{q}}^2 \tag{3.15}$$

Additionally, animals often attempt to keep their heads largely stable during their gait cycle. This is both so that the brain is not jostled too much, and because motion of the head interferes with the visual processes needed to move through an environment. The third term in the objective function is thus used to penalize translational motion of the head. Since motion of the head in the direction an animal is running does not much interfere with vision, only the component of this motion which is perpendicular the the velocity of the gait is penalized.

$$C_3 \sum_{i} \left\| \dot{p}(i, head)_{\perp} \right\|^2 \tag{3.16}$$

The fourth term functions similarly and keeps the animal's head facing in its direction of motion:

$$C_4 \sum_{i} \left\| R(i, head) - \boldsymbol{I} \right\|^2 \tag{3.17}$$

Each of these terms is weighted with a constant so that their contributions to the overall objective function are of the same general magnitude. Note that although these constants must be hand tuned, the same values tend to work for all animals and gaits: $C_1 = 25$, $C_2 = 0.1$, $C_3 = 25$, $C_4 = 100$.

3.2.4 Implementation

The general form of a spacetime constraints problem (equation 3.1), combined with the definitions for the objective function and constraint functions just given, provide a complete mathematical description of a nonlinear programming problem. Problems of this sort have been extensively studied in the optimization community, and some relatively powerful

techniques for solving them have been developed. The most efficient of these techniques, however, require the ability to efficiently computer derivatives of the objective and constraint functions.

In the case of the objective function the required derivative is generally the gradient of a scalar-valued function over an n-dimensional Cartesian space:

$$cost : \mathbb{R}^n \to \mathbb{R} \tag{3.18}$$

$$cost' : \mathbb{R}^n \to \mathbb{R}^n \tag{3.19}$$

With the constraint functions it is convenient to combine the equality and inequality constraints into a single set of doubly-bounded constraints:

$$a_i \leq c_i(M) \leq b_i \tag{3.20}$$

where $a_i \leq b_i$ and it is allowed that $a_i = -\infty$ and $b_i = \infty$. If there are m constraint functions in total, then their combination can be viewed as a single vector valued function:

$$c : \mathbb{R}^n \to \mathbb{R}^m \tag{3.21}$$

$$c' : \mathbb{R}^n \to \mathbb{R}^m \times \mathbb{R}^n \tag{3.22}$$

the derivative of which at a single point M is an $m \times n$ Jacobian matrix $\boldsymbol{J}$.

Since the most important optimization methods will require many evaluations of $cost'$ and c' during the course of the optimization, it is important that these derivatives be evaluated efficiently. In particular, at a minimum a differentiation method should utilize the fact that each variable in the range of $cost'$ often depends only on a small subset of variables in its range, and that the Jacobian matrix $\boldsymbol{J}$ is generally sparse.

The differentiation method used for the examples shown in this thesis is based on an operator overloading scheme similar to that in [21]. The constraint and objective functions are coded in Python, and operator overloading is used to construct a function composition

graph for both of these functions. Derivatives can then be analytically computed from these function composition graphs, and C++ code to compute *cost*, *cost*$'$, c, and c' is automatically generated. This approach yields relatively high-speed results and allows the sparsity pattern of $\boldsymbol{J}$ to be computed analytically. The resulting nonlinear programming problem is then solved using the SQP based nonlinear programming package SNOPT [19]. Running times for a single spacetime optimization run from around a 30 seconds for very simple animals such as a simplified biped creature to 10 minutes for more complicated ones such as a horse.

3.2.5 Results

The system described thus far can successfully solve for qualitatively plausible gaits for a variety of animals with substantially different forms. I will illustrate this here on a set of five different test creatures:

name	**# legs**	**# DOFs**	**mass**	**height** (approx.)
monoped	1	10	3.74kg	0.6m
simple-biped	2	16	10.6kg	1.2m
velociraptor	2	33	19.4kg	0.75m
horse	4	29	502kg	1.5m
pentaped	5	32	109kg	0.7m

Each of these animals additionally has a default 'rest pose' defined which serves as an initialization to the optimization. For all animals the gait cycles were synthesized with 30 frames. An illustration of these animals in their rest poses is given if figure 3.3.

In all cases the spacetime constraints optimization successfully solved for a reasonable gait given just a single static pose and the foot contact times. Sequences of images conveying each of the synthesized gaits for these animals are shown in figures 3.4, 3.5, 3.6, and 3.7.

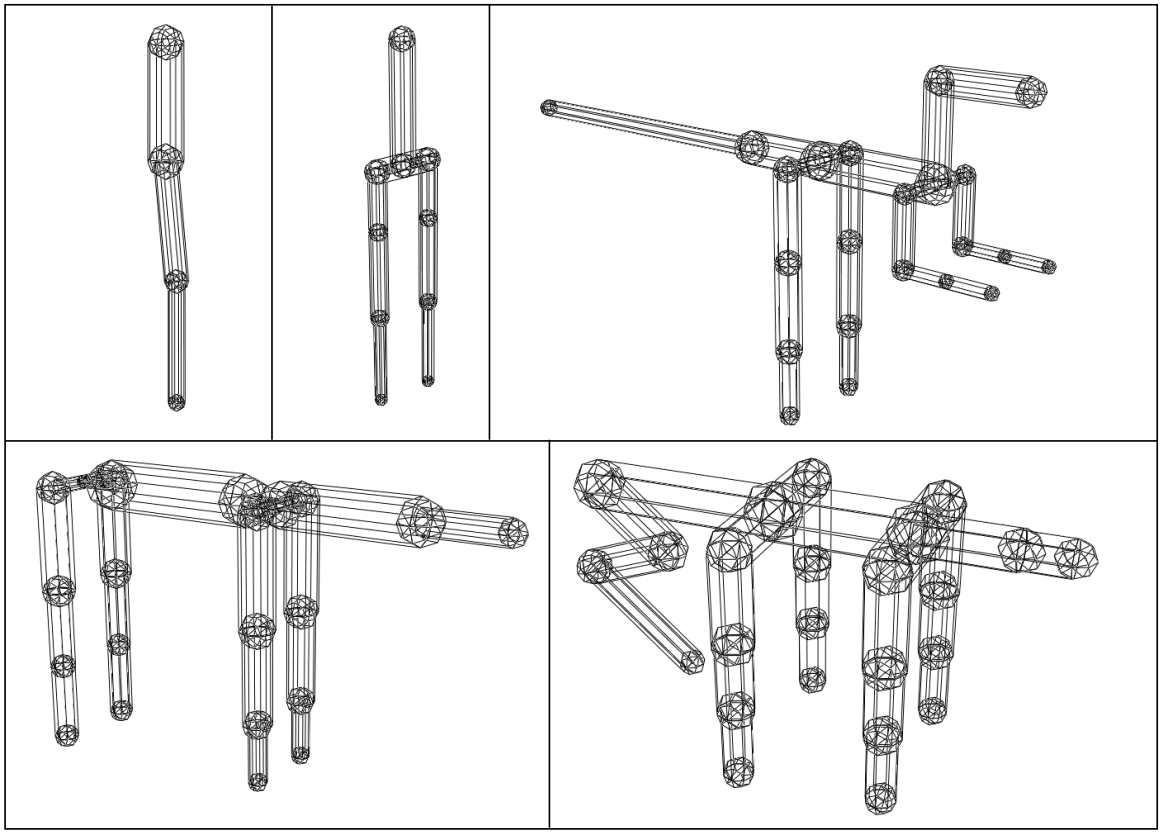


Figure 3.3: Five different creatures used to illustrate the automatic synthesis of animal gaits on animals with a wide range of different shapes. From top left to bottom right, the animals, are monoped, simple-biped, velociraptor, horse, and pentaped.

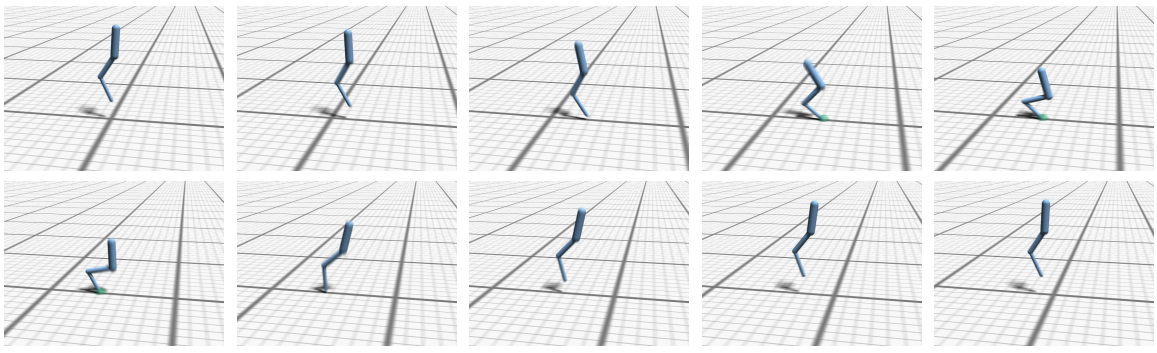


Figure 3.4: Frames from an animation of an automatically synthesized gait cycle for a simple monopedal animal.

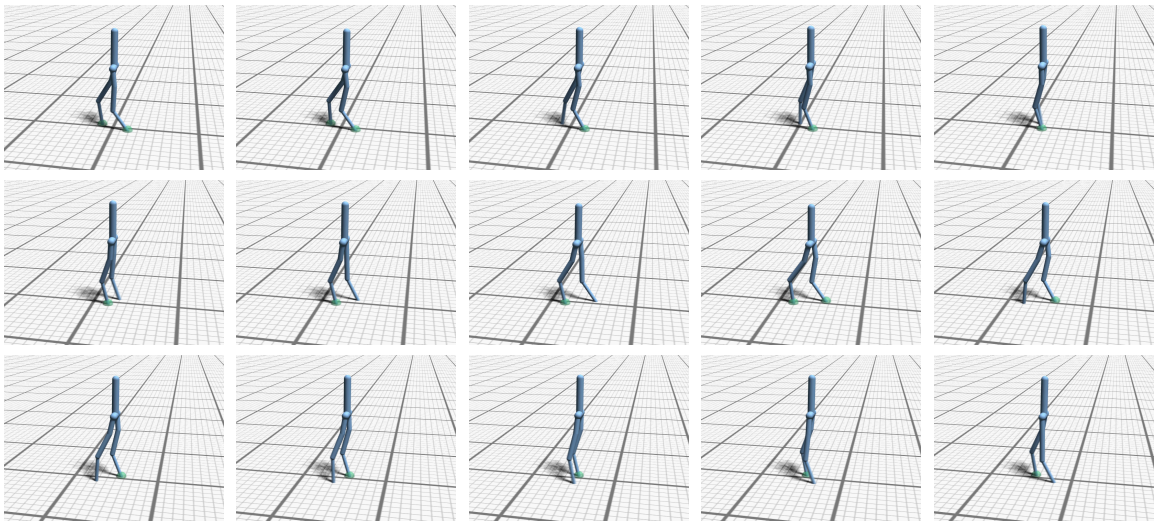


Figure 3.5: Frames from an animation of an automatically synthesized gait cycle for a simple bipedal animal. The foot contact timings and velocity have been hand-specified to describe a walking gait.

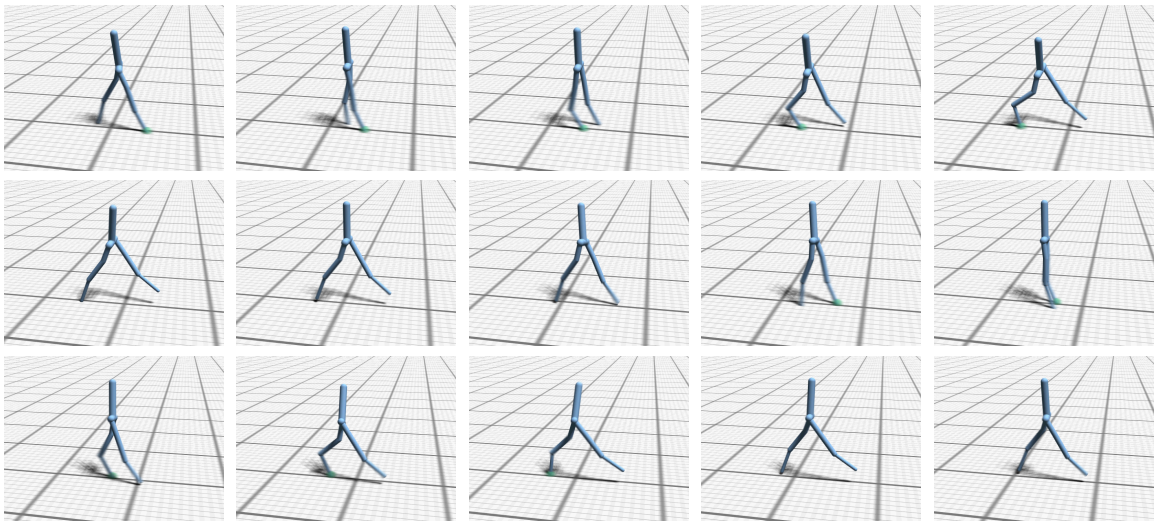


Figure 3.6: Frames from an animation of an automatically synthesized gait cycle for a simple bipedal animal. The foot contact timings and velocity have been hand-specified to describe a running gait.

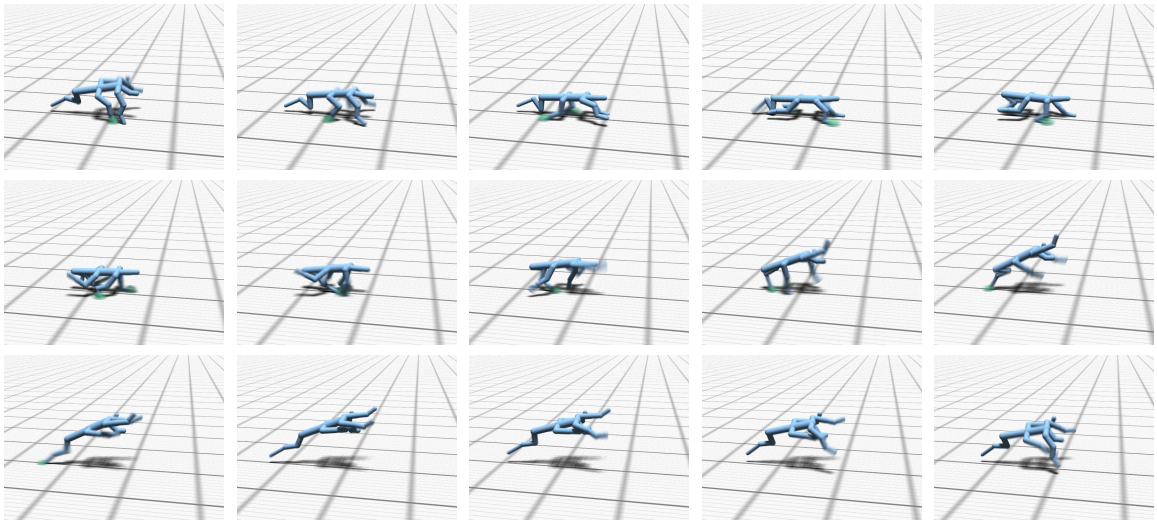


Figure 3.7: Frames from an animation of an automatically synthesized gait cycle for a fictional pentapedal animal. The foot contact timings and velocity have been hand-specified to describe a pentapedal analogue of a galloping gait.

3.3 Gait Optimization

The optimization as described so far requires that the foot contacts be specified in advance. This is a disadvantage in defining such optimizations, as it is not always obvious what the foot contact timings should be for a given animal moving at a particular speed – particularly when the animal is extinct or fictional. In this section I'll describe an optimization technique which can automatically determine the contact times for a gait. This optimization approach, however, is more general than just this and can be usefully employed in a wide range of difficult constrained optimization problems, for which it often significantly improves on the state of the art.

Since the optimization process will now be extended to include the foot contact times, it is necessary to numerically represent these contact times. To achieve this I assume that each foot touches the ground for only a single interval during a gait cycle. A gait's timing can then be represented by introducing a single variable controlling the overall period of the gait and adding a pair of optimization variables for each foot giving the start time and duration of the foot's contact interval, both measured as fractions of the gait's period.

In order to optimize over the foot contact timings it is further necessary to formulate the spacetime constraints optimization so that they can be quickly changed at runtime. Since the code to calculate derivatives is pre-generate, it is necessary to first formulate a problem that includes ground reaction forces for each foot in every frame, thus including all derivatives that might be needed later for any possible contact timing. At runtime this optimization is then modified by removing the variables and constraints corresponding to inactive foot contacts. The ground reaction force variables can be removed by forcing them to be equal to zero. The inactive constraints can be removed directly from the optimization's constraint vector and Jacobian. If automatic or numeric differentiation is used instead of symbolic differentiation then this step will probably not be necessary.

The resulting optimization is, unfortunately, more difficult to solve than the continuous optimization described in the previous section. This is because the variables for the foot contact times and durations are not differentiable, and thus standard derivative-based constrained optimization methods cannot be applied directly. Furthermore, this problem

contains many local minima which must be avoided in determining a good gait. I address both of these difficulties with a hybrid optimization scheme combining derivative aware and derivative-free techniques.

3.3.1 Hybrid optimization

In order to solve the gait optimization problem I use a novel hybrid optimization technique which combines a spacetime optimization as an inner loop with a sampling-based derivative-free optimization method based on a variant of the covariance matrix adaptation evolution strategy (CMA). This combines the efficiency in high dimensional spaces and ability to handle general constraints of spacetime optimization with the ability to handle non-differentiable variables and avoid many local minima. I will first describe the standard CMA algorithm briefly. For further details readers should consult [24] or [23].

The CMA optimization process begins with a function to be minimized, f, and an initial Gaussian distribution defined by a mean, $\mathbf{m}_0$, and a covariance matrix $\boldsymbol{C}_0$. First, λ samples are drawn from this distribution and f is evaluated at each. After this is done the μ samples with the lowest associated values of f are selected. These samples are termed the *elites* and will be denoted by $\mathbf{x}_1, \ldots, \mathbf{x}_\mu$ where $f(\mathbf{x}_1) \leq \cdots \leq f(\mathbf{x}_\mu)$. Each elite is also associated with a weight as defined by:

$$w_j = \frac{\ln(\mu + 1) - \ln(j)}{\mu \ln(\mu + 1) - \sum_{k=1}^{\mu} \ln(k)} \tag{3.23}$$

These weights are chosen so as to favor those elites with the lowest values of f, and other weighting schemes may also work, so long as $w_1 > \cdots > w_\mu > 0$. Using these values for w and $\mathbf{x}$, the mean is updated as:

$$m_{i+1} = \sum_{j=1}^{\mu} w_j \mathbf{x}_j \tag{3.24}$$

and similarly update the covariance matrix as:

$$\boldsymbol{C}_{i+1} = (1 - c_{cov})\boldsymbol{C}_i + c_{cov} \sum_{j=1}^{\mu} w_j (\mathbf{x}_j - \mathbf{m}_j)(\mathbf{x}_j - \mathbf{m}_j)^{\mathbf{T}} \tag{3.25}$$

As this method is iterated the multivariate Gaussian moves and shrinks until it becomes a point at the function's minimum or until a preset maximum number of iterations is reached.

3.3.2 Basin-CMA

The CMA algorithm has a few disadvantages which make it unsuitable for direct use in gait optimization. Firstly it does not take advantage of derivatives when they are available, and thus is somewhat inefficient on the largely differentiable gait synthesis problem. More importantly, however, CMA is an entirely *unconstrained* optimization approach so it is unable to handle the kinematic and dynamical constraints in the spacetime constraints problem.

The solution to both of these problems is to use a spacetime constraints optimization at each CMA sample point instead of evaluating the objective function directly. More formally, let f be the objective function and $c_1, \ldots, c_n$ be the constraint functions. Furthermore, let g be a function such that if $\mathbf{y} = g(\mathbf{x}|f, c1, \ldots, c_n)$ then $\mathbf{y}$ is a local minimizer of f which satisfies the constraints $c_1, \ldots, c_n$ where the minimization is started from the initial point $\mathbf{x}$. Generally, g can be thought of as projecting $\mathbf{x}$ to the nearest constrained local minimum.

The CMA algorithm can then be modified so that it searches over $f \circ g(\mathbf{x}|f, c1, \ldots, c_n)$ instead of $f(x)$. As this search is essentially over the function defined by the basins of attraction for $g(\cdot|f, c1, \ldots, c_n)$, I term this optimization basin-CMA. This allows basin-CMA to leverage the local optimization's constraint handling behavior to quickly adapt to the constraint manifold. An illustration of this can be seen in figures 3.8 and 3.9.

In addition to changing the objective function over which optimization is searching, Significantly greater efficiency can be achieved by harmoniously altering the update equations. Given the elite sample points $\mathbf{x}_1, \ldots, \mathbf{x}_\mu$, define $\mathbf{y}_j = g(\mathbf{x}_j|f, c1, \ldots, c_n)$ for $1 \leq j \leq \mu$. Then both the mean and covariance update equations (3.24, 3.25) are altered by replacing each use of $\mathbf{x}_j$ with $\mathbf{y}_j$.

Although this change is very simple, the resulting optimization is, in my experience, both surprisingly efficient and quite powerful. I have tested this method against the problems from the 2006 Congress on Evolutionary Computation (CEC-06) real-parameter constrained

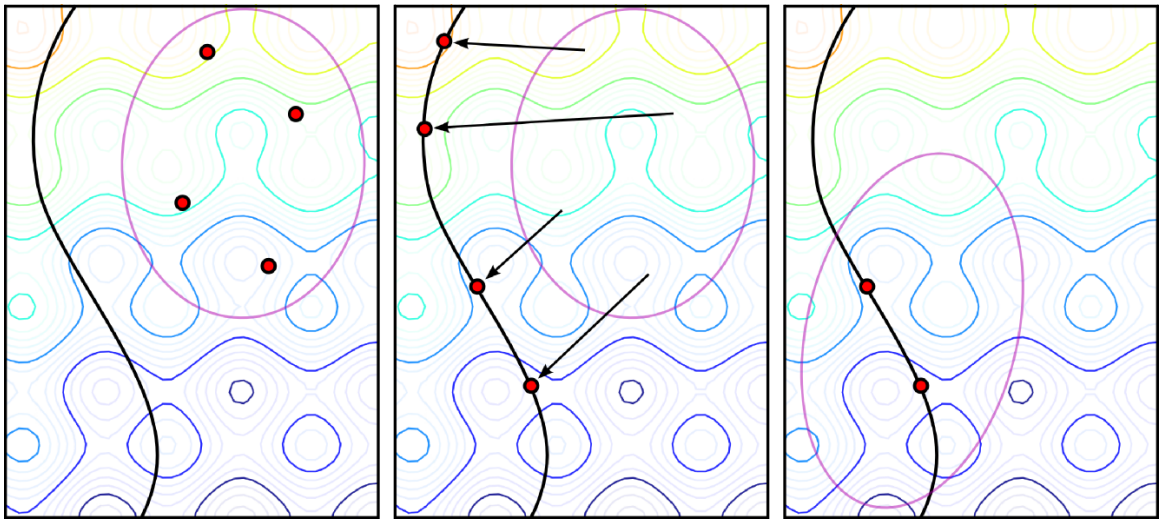


Figure 3.8: An illustration of a single iteration of basin-CMA. First λ samples are chosen form the current distribution. Next each sample is projected to a local constrained minimum. Finally the mean and covariance are updated according to the elite samples.

optimization competition. The problem set consists of 24 non-convex constrained optimization problems ranging from 2 to 24 dimensions, all of which exhibit local minima. I found that basin-CMA was able to match the ability to find the global constrained optimum of the best of the other entrants while running faster than the competition's best entrant for each problem by an average factor of 51 times if derivative evaluations are as cheap as function evaluations and 6.3 times if finite differences are used for all derivatives. Full details of these tests are given in appendix B.

3.3.3 Application to gait optimization

The basin-CMA procedure is used as the outer loop in the optimization and employed in handling non-differentiable or poorly conditioned terms such as the foot contact timings. In doing so, however, it is necessary to define the local optimization function, $g(\mathbf{x}_j|f, c1, \ldots, c_n)$ for $1 \leq j \leq \mu$, appropriately so that the variables dictating the contact times are not passed to the continuous spacetime constraints optimization, which has no means to handle them.

This is done in the definition of g by labeling each variable in the optimization with one

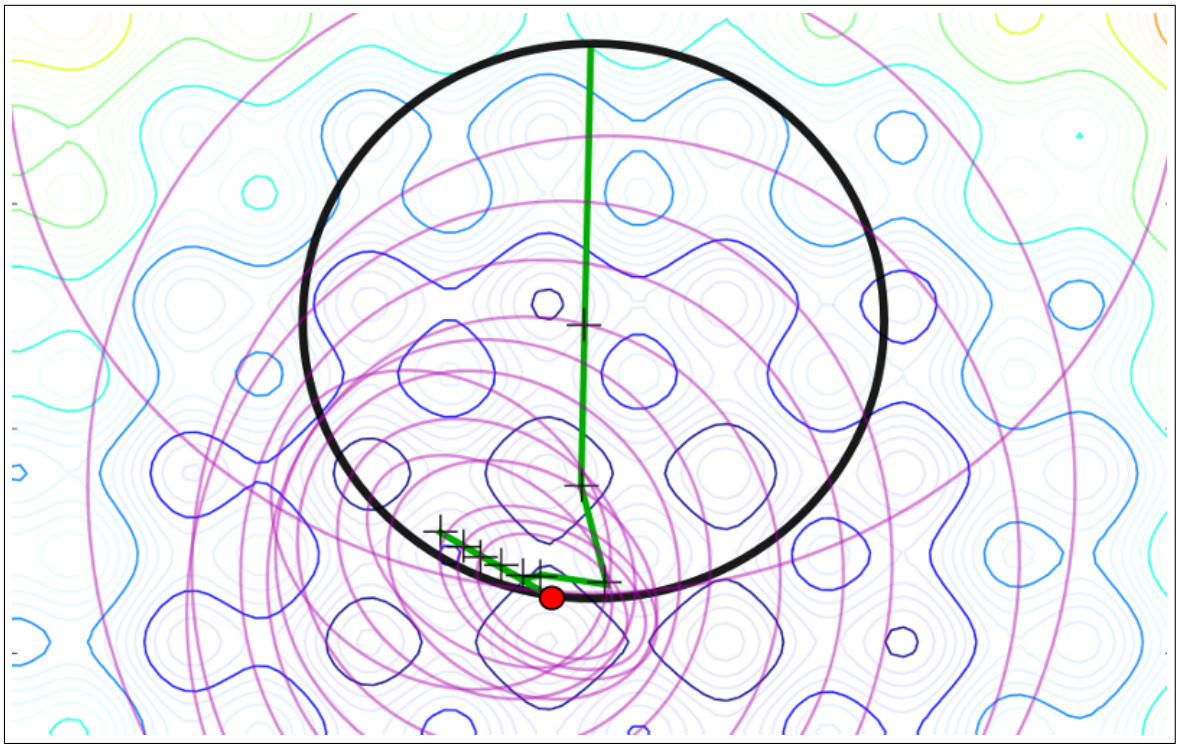


Figure 3.9: The progress of basin-CMA on finding the global optimum of a constrained minimization problem. The constraints enforce the solution to be on the unit circle, shown in black. The means follow a path displayed in green while the covariance matrices at each iteration are drawn in purple. Towards the end of the optimization the covariance matrix can be seen adapting to the constraint manifold.

of three labels dictating its use as excluded from the local optimization, included in both the CMA and local optimizations, or used in the local optimization only. This labeling gives flexibility both in excluding variables from the local optimization which it cannot handle properly, and in reducing the dimensionality of the problem solved by the basin-CMA outer loop in the case that the local optimization can handle some variables well when given only their default values. Typically only the foot timings are included in CMA while excluding the pose, passive element, and ground force variables (as well as the morphology variables to be introduced later). This means that the outer loop normally only has to optimize over three to ten dimensions. If needed, however, the pose and morphology variables can be managed in the outer loop as well, although the resulting basin-CMA optimization will

generally have several hundred dimensions and converge somewhat more slowly, although the resulting optimization is highly robust to poor initialization, a feature quite atypical of spacetime constraints optimizers.

In order to initialize the optimization I choose a Gaussian distribution wide enough so that it covers the entire allowed parameter space of the CMA variables. More specifically the mean is set to a default in which all variables are zero except for the foot timings, which are set to be evenly spaced within a one-second period gait, and the torso height which is set so that the feet are level with the ground. The covariance matrix is diagonal with each element being the maximum distance from the mean to one of the two bounds in the corresponding dimension (or 500 if the dimension is partially or fully unbounded). In the local optimization the variables not dictated by CMA (i.e. marked as "local optimization only") are initialized to their default values. So long as the initial CMA distribution is wide enough to provide good coverage of the allowed parameter space many other initialization schemes would give equivalent results.

3.3.4 Results

The basin-CMA based gait optimization is successful in finding foot contact timings for all of the test animals (see figure 3.3). This includes automatic determination of a walk/run gait depending on the desired speed, as well as picking gaits which effectively balance the animal's mass between the feet of both the quadruped and pentaped animals. Examples of these foot timings are shown in figure 3.10. In the case of the fictional pentaped animal, the gait automatically determined with this optimization is entirely novel and was substantially more efficient than a hand-designed gait (based on a gallop). In addition, multiple runs of basin-CMA generally converge to similar gaits, or to one of a set of reasonable gaits, for instance running versus hopping for a biped moving at high speed. Visual depictions of the gaits resulting from these optimizations can be seen in figures 3.11, 3.12, 3.13, 3.14, and 3.15.

My approach does not yet fully capture all the details exhibited in animal motions, and was not not able to automatically obtain, for example, a gallop gait for the horse – getting

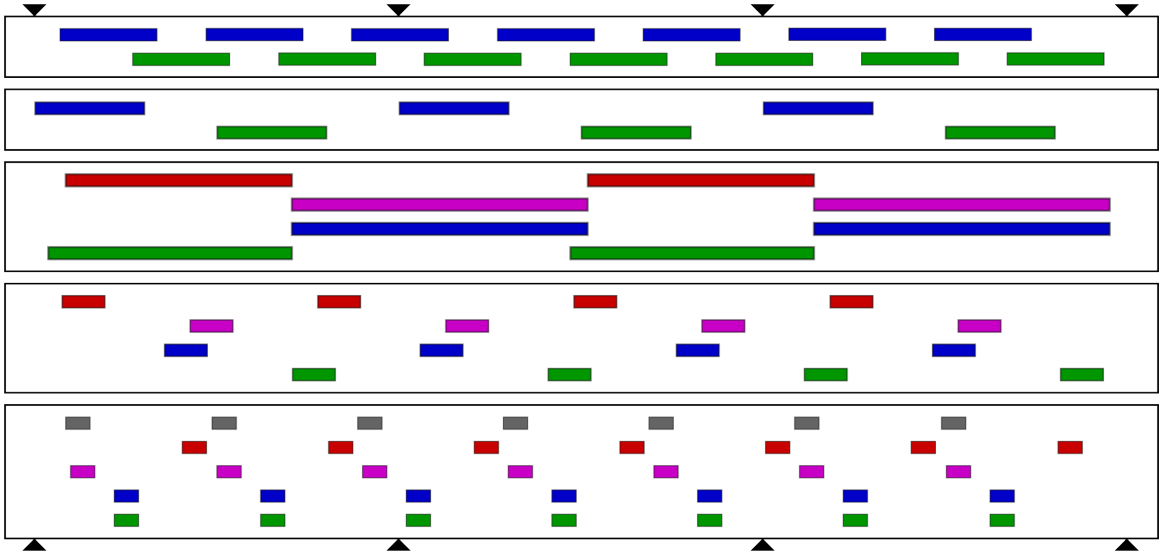


Figure 3.10: Examples of the foot contact timings resulting from the basin-CMA optimization method. From top to bottom: simplified biped at $0.7\frac{m}{s}$, simplified biped at $3.0\frac{m}{s}$. horse at $1.0\frac{m}{s}$, horse at $10.0\frac{m}{s}$, and pentaped at $4.0\frac{m}{s}$.

instead a four-beat phase shifted trot (figure 3.10, second from bottom). I suspect that this is due to an incomplete modeling of biomechanical elements such as an animal's preference for using some muscles over others, robustness, and complexity of control. Nevertheless my method can easily be used to optimize over a reduced subspace of all possible foot timings, so an artist can design some aspects of a gait they desire and our method can search within this for an optimal gait.

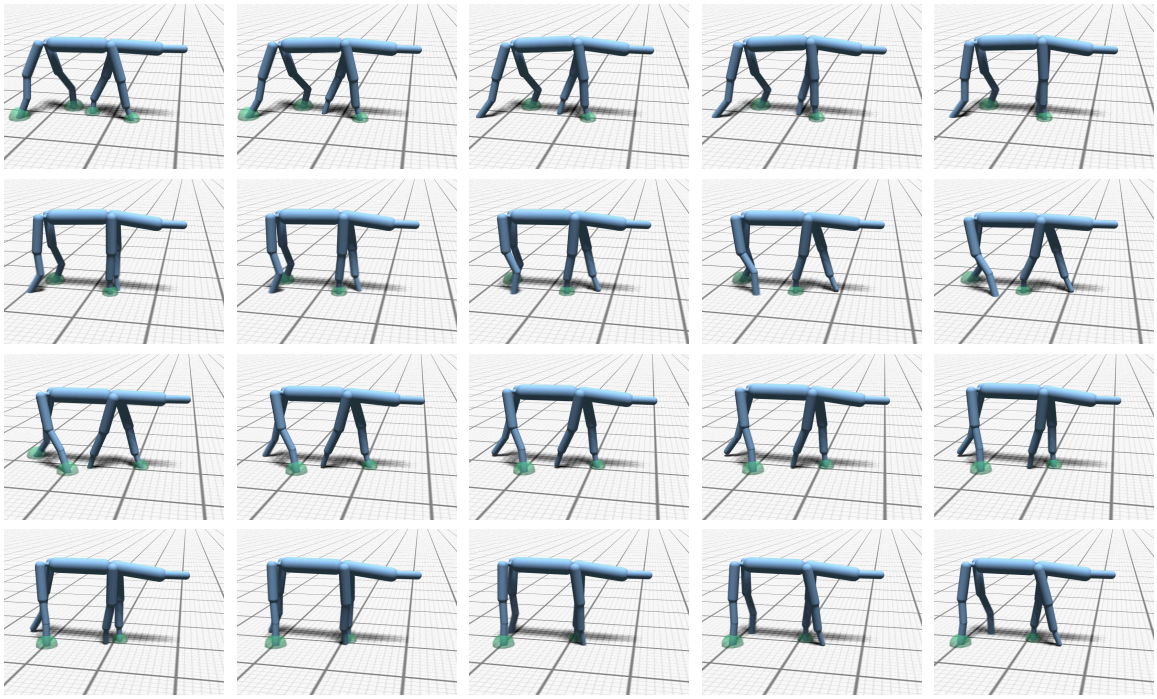


Figure 3.11: Frames from an animation of an automatically synthesized gait cycle for a horse-like quadrupedal animal moving at a speed of $1\frac{m}{s}$. The foot contact timings have been automatically solved for, and describe a walking-trot.

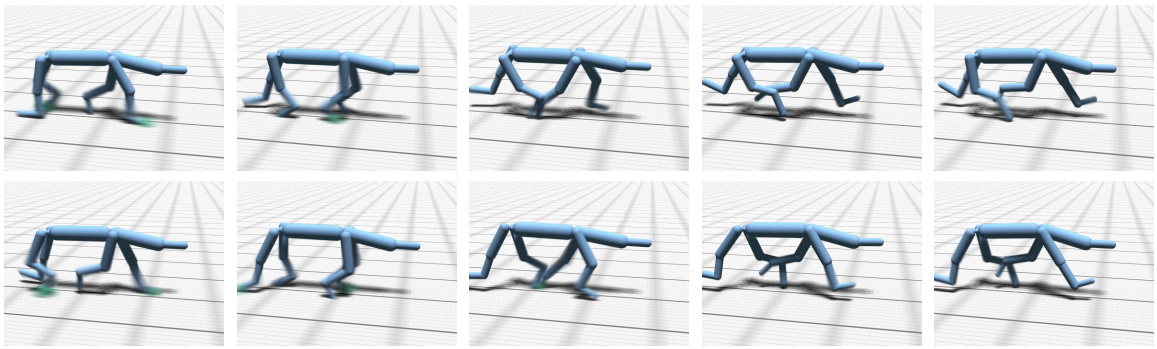


Figure 3.12: Frames from an animation of an automatically synthesized gait cycle for a horse-like quadrupedal animal moving at a speed of $10\frac{m}{s}$. The foot contact timings have been automatically solved for, and describe a running-trot with a phase offset between the diagonal pairs of legs.

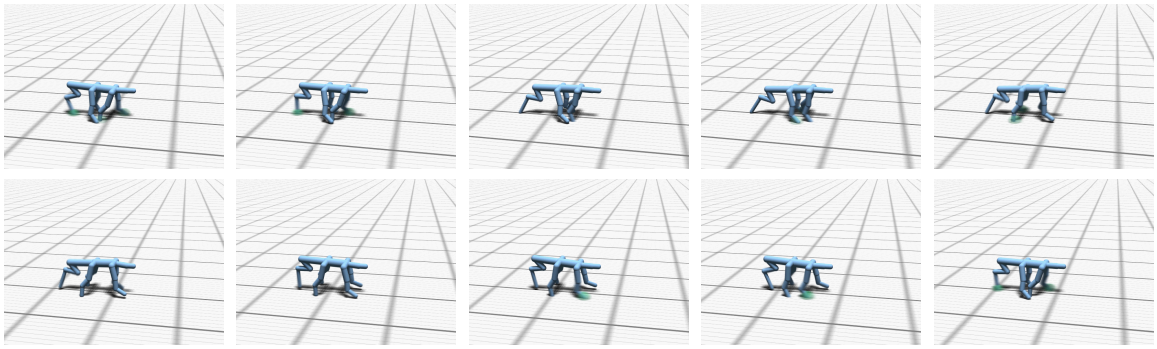


Figure 3.13: Frames from an animation of an automatically synthesized gait cycle for a pentapedal animal moving at a speed of $3\frac{m}{s}$. The foot contact timings have been automatically solved for, and describe an entirely novel gait which alternates balancing the animal's weight between the outer and inner legs.

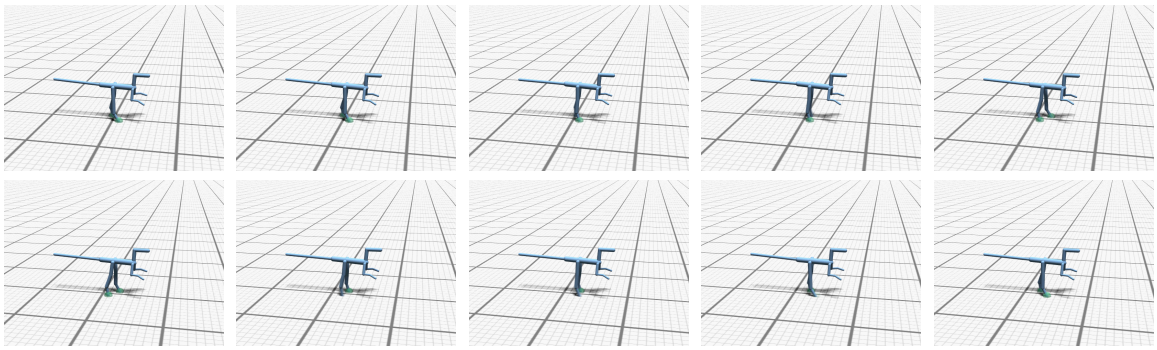


Figure 3.14: Frames from an animation of an automatically synthesized gait cycle for a bipedal animal resembling a velociraptor moving at a speed of $0.7\frac{m}{s}$. The foot contact timings have been automatically solved for, and describe a walking gait.

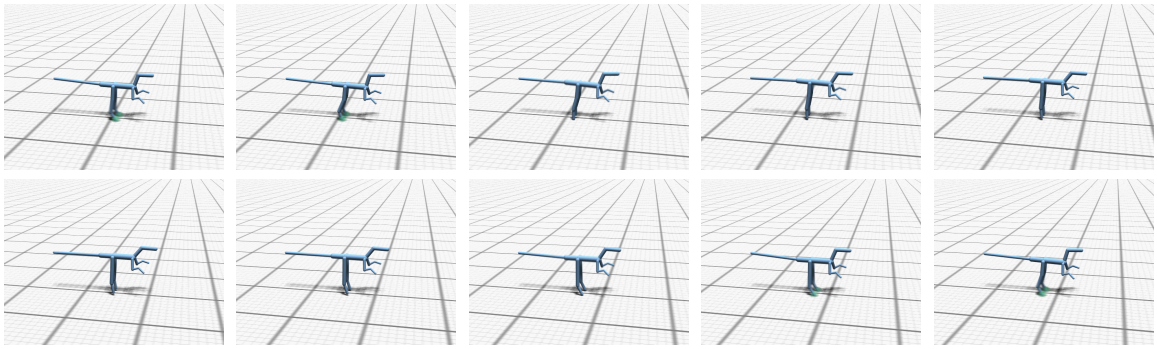


Figure 3.15: Frames from an animation of an automatically synthesized gait cycle for a bipedal animal resembling a velociraptor moving at a speed of $2.5\frac{m}{s}$. The foot contact timings have been automatically solved for, and describe a hopping gait.

3.4 Morphology Optimization

The core continuous spacetime constraints optimization described in section 3.2 follows closely in the mold of previous approaches and is sufficient for generating gaits on an animal with a pre-specified shape. A somewhat surprising and useful extension of this method is to not only solve for an animal's gait, but also refine the shape of the animal itself to better perform some user-specified task. This not only better approximates reality, in which an animals form and gait both co-evolve together, but has useful applications in computer graphics as well. A gait that looks 'wrong' is sometimes the result of an improper specification of a creature's shape, and by allowing this shape to be automatically adjusted the resulting gaits often appear more natural.

There are relatively few methods which address this optimization over a creatures shape as well as its motion. Probably the best known approach which does tackle this problem, however, is Karl Sim's *Evolving Virtual Creatures* [68]. This paper gives a genetic algorithm to search for both a shape and a control strategy which would allow a creature to best perform a pre-specified task, such as swimming or moving on land. The resulting animations are highly entertaining and often lifelike, but rarely resemble those of actual animals in the case of terrestrial locomotion.

My method for finding an animal's gait and its shape is similar in spirit to that of Sims' and others in that it is based upon an optimization which simultaneously solves for both form and motion, but the specifics of how this is achieved are different. Instead of optimizing over completely different skeleton topologies, only the lengths and radii of an animal's limbs are varied. This restriction allows the use efficient derivative-based optimization methods to solve for shapes and motions more closely resembling those found in nature. Although this means that for instance an extra leg cannot automatically added to an animal, in practice the range of possible shapes is still large – a dog and a horse have the same topology.

In order to allow optimization over morphology it is necessary to add two new optimization variables per limb controlling the radius and length of the limb, scaling the mass of the bone proportionally so that it maintains a constant density. Thus larger limbs, while often providing an advantage in terms of locomotion, also carry the disadvantage that they

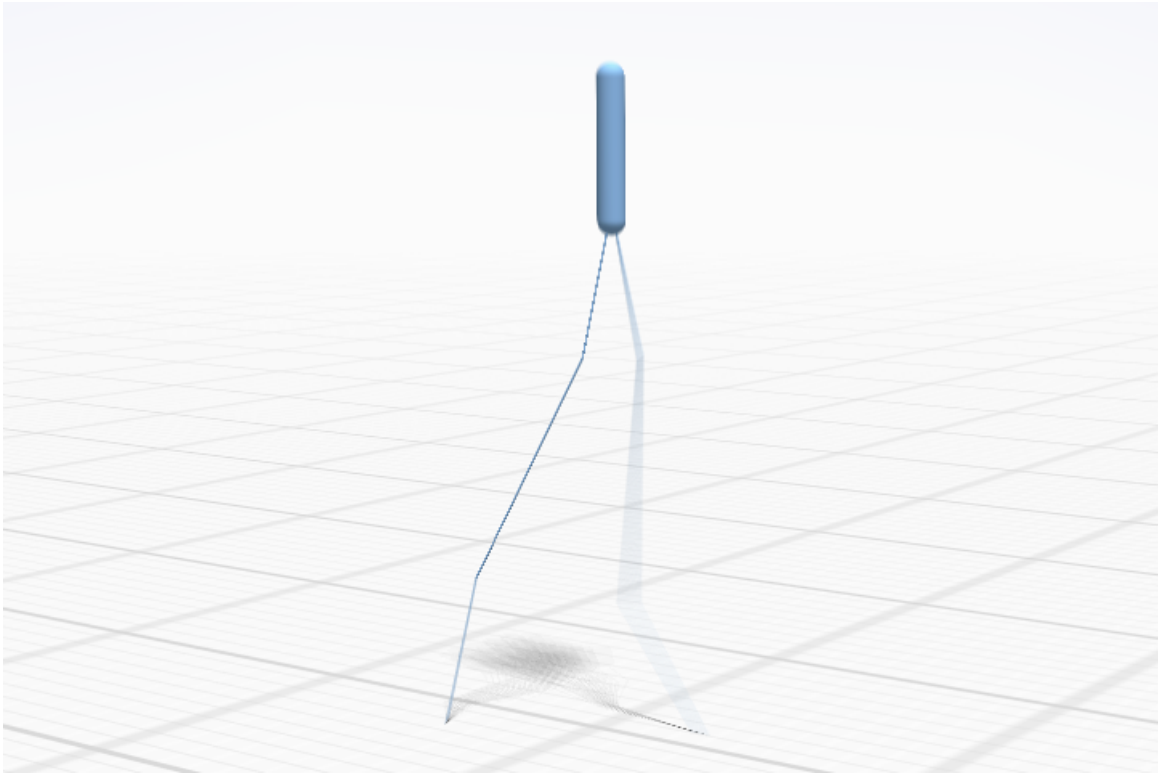


Figure 3.16: An example of the morphology generated for a simple bipedal creature using the spacetime constraints optimization as previously specified in section 3.2. Note how the optimization is able to "cheat" by making the creature's legs impossibly long and slender. The modifications described in section 3.4.1 penalize these sorts of physically unrealizable animal shapes.

increase the animal's mass. These variables are defined once over the entire gait, and do not vary from frame to frame like the pose of ground reaction force variables.

At first glance, it might seem that optimizing over an animal's shape in this manner would be just as simple as adding the bone length and radius variables to the optimization. It turns out, unfortunately, that this does not work nearly so well as one might hope, as the objective function defined in section 3.2.3 allows the optimizer to "cheat" in ways that aren't biologically realistic. For example, figure 3.16 shows the result of running a standard spacetime constraints optimization as previously described to co-optimize for the motion and form of a simplified bipedal creature. Since there is nothing the formulation to penalize

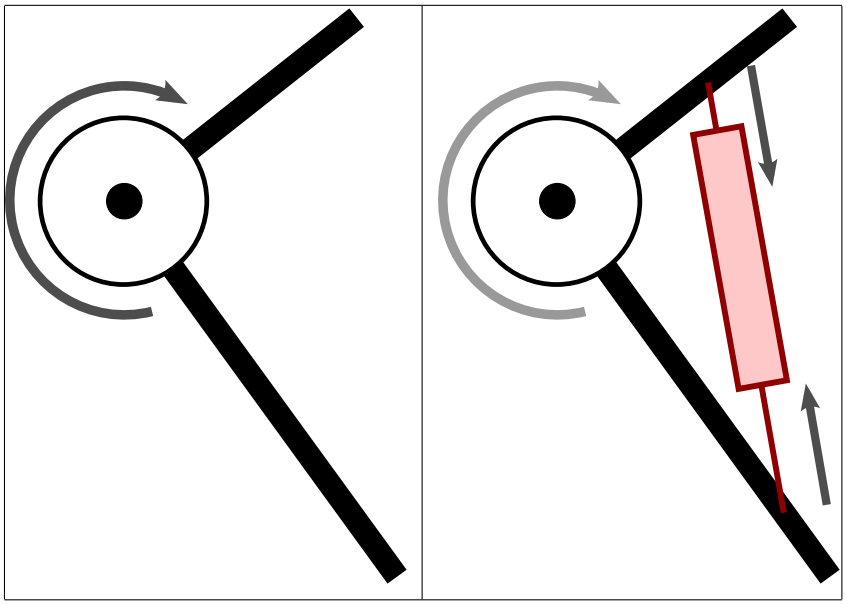


Figure 3.17: Two different ways of modeling the torques necessary at each joint to perform a motion. On the left is the standard method in which joint torques are applied directly. On the right is the approach used when optimizing over an animal shape where the forces arise only by virtue of the force from a muscle attached to the limbs adjacent to the joint.

it for doing so, the optimization is able to make the biped's legs impossibly long and slender, thereby decreasing the animal's mass and the required joint torques. The next section will describe modifications to the cost function which avoid these sorts of unrealistic animal shapes.

3.4.1 Morphology-Aware Objective Function

The key problem with attempting to co-optimize over an animal's form and gait by adding optimization variables for each limb's length and radius and scaling the limb's mass appropriately is that there are other ways in which a limb's size impacts the biological feasibility of a gait than just by its mass. In particular, thinner limbs can only support thinner muscles, making the associated limbs weaker (see figure 3.17).

I approximate this relationship between a limb size and muscle strength by altering the objective function given in section 3.2.3 to better reflect the tradeoffs of having larger versus smaller limbs. This is achieved by redefining the torque minimization term in equation 3.14,

replacing it with a term attempting to minimize the active forces exerted by the muscles, E_f, and a term attempting to minimize the stress put upon the muscle fibers, E_σ:

$$C_1 \left[\sum_{i,j} E_f(i,j)^2 + \sum_{i,j} E_\sigma(i,j)^2 \right] \tag{3.26}$$

In order to calculate E_f and E_σ an estimate must be made at each actuated joint, j, of both the cross-sectional muscle areas of those muscles controlling the joint, denoted a_j, and a scaling factor for converting muscle forces into joint torques, denoted d_j. In my approach only the basic shape of the animal is given by the optimization variables and instead few simple heuristics are used to guess the values of a_j and d_j. This has the advantage of simplifying the optimization but has the disadvantage that the resulting model is a rather crude biomechanical approximation of reality. Nevertheless this approach has proved sufficient for qualitatively reasonable results in solving for an animal's shape.

The cross-sectional muscle area, a_j, for each joint, j, is estimated by assuming that 25% of a limb's cross-sectional area is taken up by muscle. Furthermore, generally (although certainly not always) in nature large muscles are connected at both ends to large limbs, partially because large muscle loads may damage the bones in a small limb. These properties are represented with the equation:

$$a_j = 0.25\, \pi\, smin(r_{j,1}, r_{j,2})^2 \tag{3.27}$$

where $r_{j,1}$ and $r_{j,2}$ are the radii of the limbs adjacent to joint j and $smin$ is a smooth and differentiable approximation to the min function in the non-negative quadrant:

$$smin(a,b) = \left(a^{-10} + b^{-10}\right)^{-\frac{1}{10}} \tag{3.28}$$

To estimate d_j a value is first chosen for the muscle's attachment distance on each of the two limbs at joint j, $d_{j,\{1,2\}}$, and then these are combined into a single scaling factor. $d_{j,\{1,2\}}$ is estimated by noting that the attachment arm is constrained both by the length and the radius of each limb. This is mathematically represented by setting each limb's attachment arm to be a minimum of a proportion of the length and radius of the respective limb:

$$d_{j,\{1,2\}} = smin(0.2\ l_{\{1,2\}}, 1.5\ r_{\{2,1\}}) \tag{3.29}$$

The most accurate way to combine $d_{j,1}$ and $d_{j,2}$ into a single scaling factor would include the dependence on the angle of the joint's extension. For simplicity this is approximated with a more basic approach by just deriving the force-to-torque conversion factor for when the limb is extended at a right angle:

$$d_j = \frac{d_{j,1}\, d_{j,2}}{\sqrt{d_{j,1}^2 + d_{j,2}^2}} \tag{3.30}$$

Having computed values for a_j and d_j it is now possible to define $E_f(i,j)$ and $E_\sigma(i,j)$ as used in equation 3.26. The definition of $E_f(i,j)$ is simple and merely computes the force exerted by the muscle at joint j in frame i:

$$E_f(i,j) = \frac{\left|\mathbf{t_a}_{i,j}\right|}{d_j} \tag{3.31}$$

The definition of E_σ is moderately more involved. I begin by estimating the magnitude of the force exerted upon the muscles attached to joint j, $\left|\mathbf{f_m}_{i,j}\right|$, which differs from $E_f(i,j)$ in that it accounts for forces due to both active and passive actuation:

$$\left|\mathbf{f_m}_{i,j}\right| = \frac{1}{d_j}\left(\left|\mathbf{t_a}_{i,j}\right| + \left|\mathbf{t_p}_{i,j}\right| + \left|\mathbf{t_e}_{i,j}\right|\right) \tag{3.32}$$

In addition a third term, $\left|\mathbf{t_e}_{i,j}\right|$, in included in computing this force. This prevents the optimization from allowing some limbs to become very thin by keeping all forces parallel to the limb's axis and eliminating any torques at the adjacent joints. Due to inevitable errors in a real animal's motion such precise forces are impossible to actually achieve. Thus it is assumed that it is possible for any force through a limb to actually be exerted slightly more off-axis than determined by $\mathbf{f}_{i,j}$. The magnitude of the extra torque from this displacement is calculated by $|\mathbf{t_e}|_{i,j} = 0.05\ |\mathbf{f}_{i,j} \cdot v_{limb}|$ where v_{limb} is the vector form one endpoint of the limb to the other.

Is is now possible to compute $E_\sigma(i,j)$ as a scaled multiple of $\left|\mathbf{f_m}_{i,j}\right|$:

$$E_\sigma(i,j) = h(\sigma_{i,j}) \left|\mathbf{f_m}_{i,j}\right| \tag{3.33}$$

The scaling factor, $h(\sigma_{i,j})$, is chosen so that this term starts to dominate $E_\mathbf{f}(i,j)$ when the stress upon the muscles at joint j, $\sigma_{i,j}$, reaches some value, σ_{max}. This stress is computed as the force per cross-sectional muscle area: $\sigma_{i,j} = \left|\mathbf{f_m}_{i,j}\right| / a_j$. In my case I use $h(\sigma) = \sqrt{3(\sigma/\sigma_{max})^2 + 1} - 1$, and choose a value of $\sigma_{max} = 50\frac{N}{cm^2}$, which corresponds to the maximum muscle stress exerted by endurance-trained human athletes [22].

3.4.2 Results

The approach I have described for co-optimizing over both the form and the gait of an input animal is successful in generating plausible morphologies for a range of situations, and avoids the "degenerate morphologies" such as those illustrated in figure 3.16. In the following examples, when optimizing for morphology I keep the torso, neck, and head size of the animals fixed and allow the proportions of the other limbs to vary. The morphological degrees of freedom are parametrized to preserve left-right symmetry, but no symmetry is enforced on the gait itself.

The optimization successfully adapts the morphology to tasks such as moving at different speeds, maintaining different gaits, and keeping the head above a certain height, as illustrated in 3.18. For detailed image sequences illustrating the motion of these gaits are shown in figures 3.19, 3.20, 3.21, 3.22, 3.23, and 3.24.ts the morphology to tasks such as moving at different speeds, maintaining different gaits, and keeping the head above a certain height, as illustrated in 3.18. For detailed image sequences illustrating the motion of these gaits are shown in figures 3.19, 3.20, 3.21, 3.22, 3.23, and 3.24.

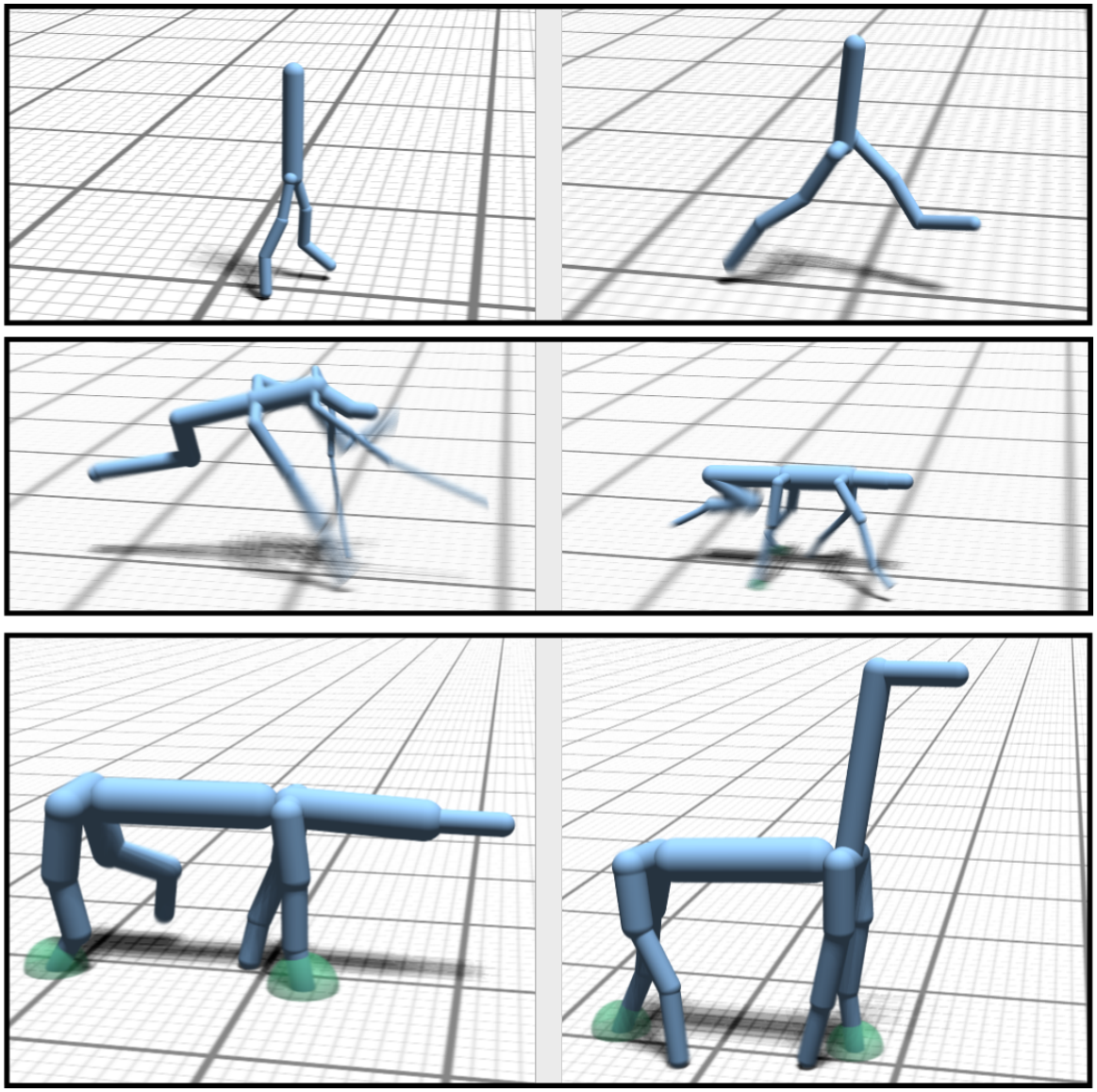


Figure 3.18: Some examples showing varying morphologies. From top to bottom: same contact times but different speeds, same speed but different contact times, and a user constraint setting the minimum height of the head. More detailed illustrations of the gaits here are shown in figures 3.19, 3.20, 3.21, 3.22, 3.23, and 3.24.

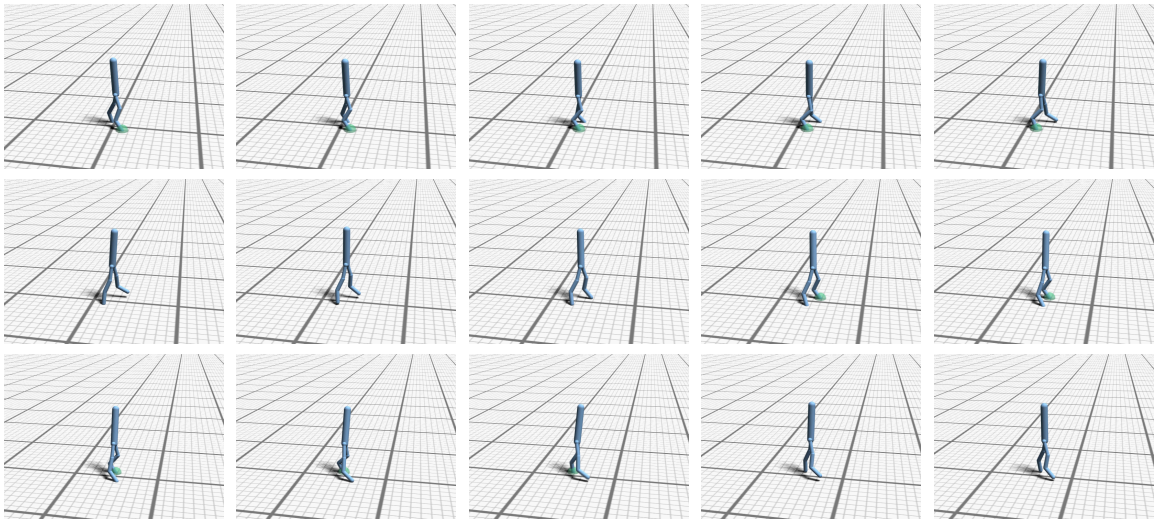


Figure 3.19: Frames from an animation of an automatically co-optimized gait cycle and morphology for a simple bipedal animal. The foot contact timings are pre-specified for a running motion, but the speed of the gait has been fixed to that of a walk. The biped's legs have been shortened to account to make a running gait more natural at this low speed.

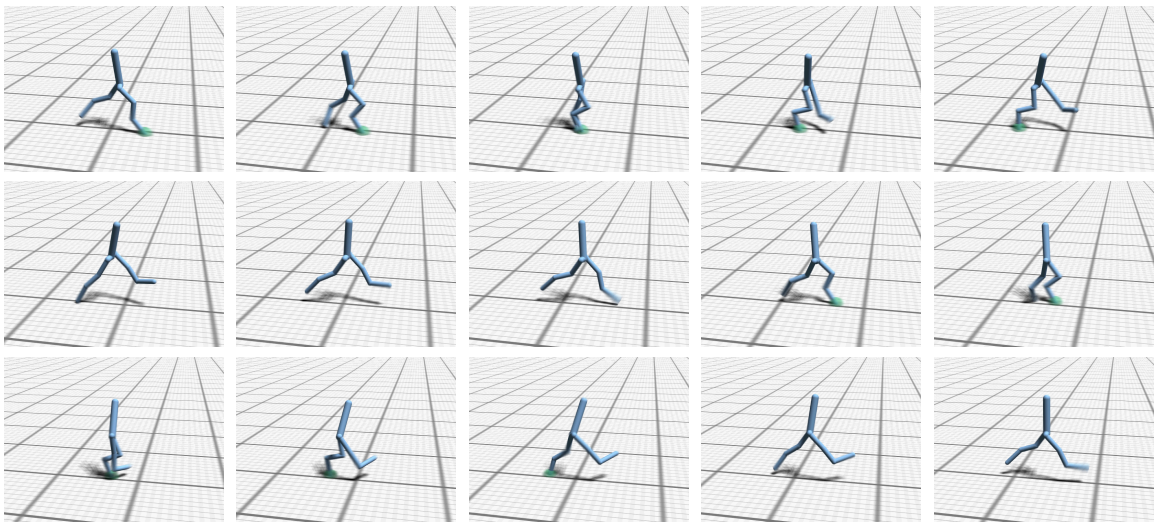


Figure 3.20: Frames from an animation of an automatically co-optimized gait cycle and morphology for a simple bipedal animal. The foot contact timings and speed are pre-specified as in figure 3.6, but by allowing optimization over morphology the resulting motion appears smoother and more natural.

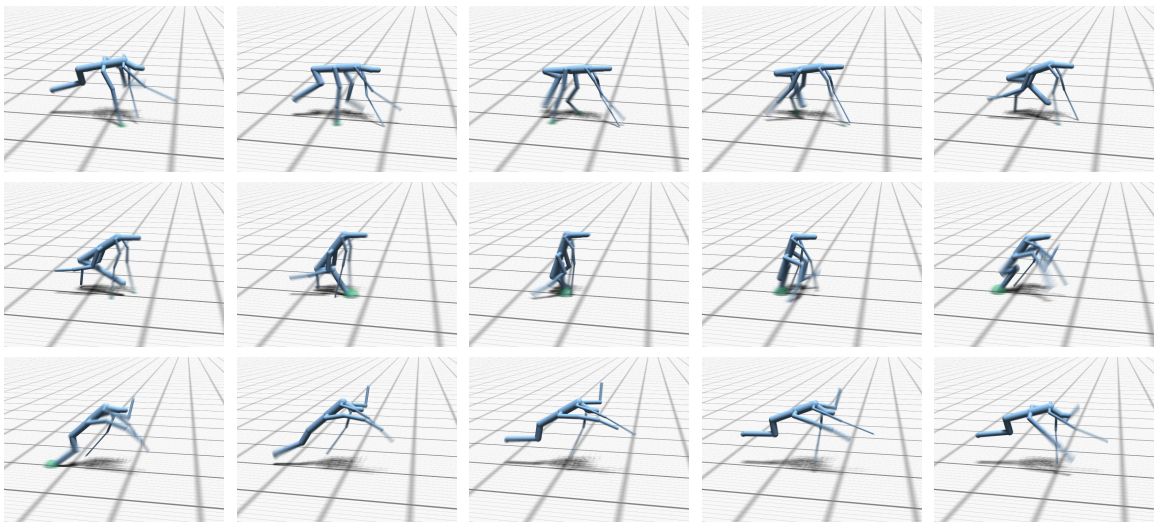


Figure 3.21: Frames from an animation of an automatically co-optimized gait cycle and morphology for a pentapedal animal. The foot contact timings and speed are pre-specified as in figure 3.7, but by allowing optimization over morphology the resulting motion appears smoother and more natural, while simultaneously highlighting that the hand-authored galloping gait is was relatively unnatural for the animal's original morphology.

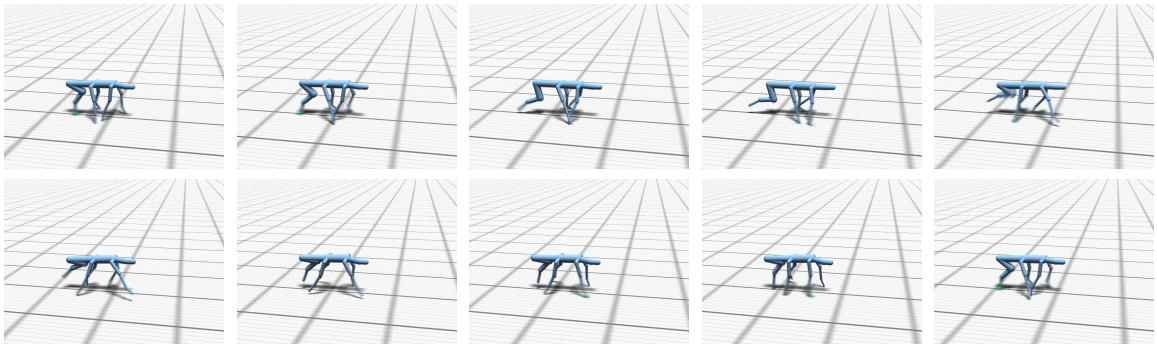


Figure 3.22: Frames from an animation of an automatically co-optimized gait cycle and morphology for a pentapedal animal moving at a speed of $3\frac{m}{s}$. The foot contact timings have been automatically solved for, and a optimal morphology generated for this novel gait cycle. It is clear how the optimized gait cycle is a much more natural fit for the animal's morphology.

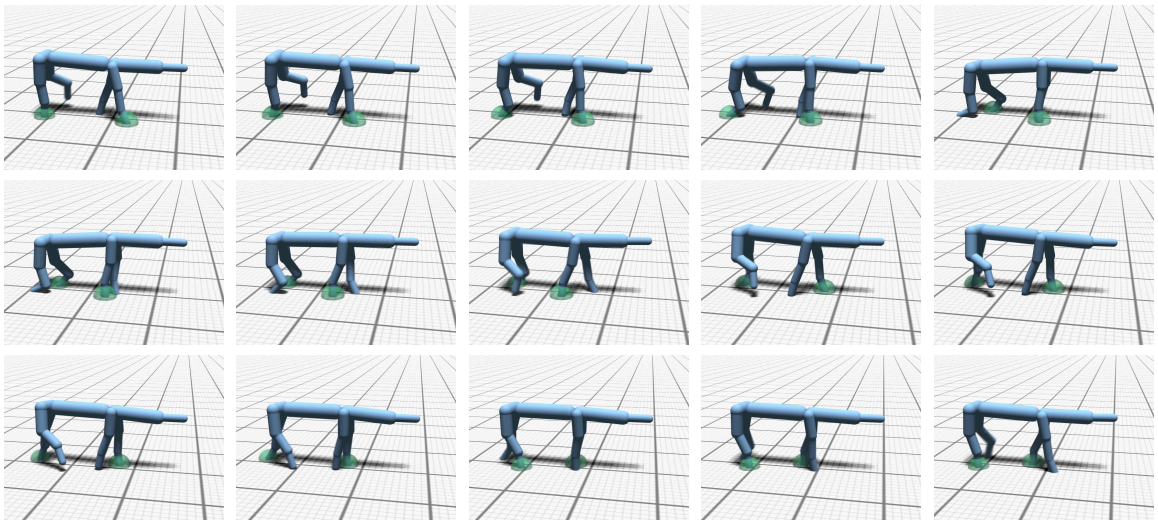


Figure 3.23: Frames from an animation of an automatically synthesized motion and morphology for a horse-like quadrupedal animal. The foot contact timings and speed are prespecified to match that of a walk. Note that to reduce the mass of the animal and move more efficiently at a walking pace the height of the quadruped has been decreased.

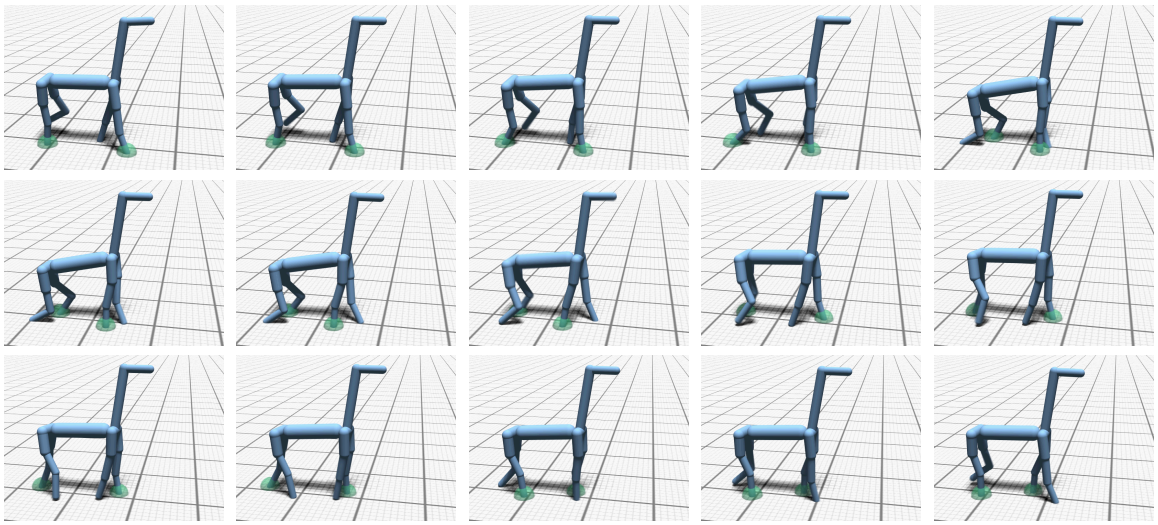


Figure 3.24: Frames from an animation of an automatically synthesized motion and morphology for a horse-like quadrupedal animal. The foot contact timings and speed are prespecified to match that of a walk. In addition, another constraint has been added enforcing a minimum height of the head. The lengthening of the neck (as opposed to the legs) is a result of achieving this head-height constraints while simultaneously minimizing the joint torques required for the motion.

Chapter 4

ACCURATE MOTION SYNTHESIS

The preceding chapter focused on the problem of reliably generating a motion given an animal and a cost function describing the particular way in which the resulting motion should be considered 'optimal'. This provides the capability to synthesize gaits which are qualitatively similar to those found in nature, but the motions are unlikely to match those of the model's real-world counterparts to a particularly high degree of fidelity. The second component to this thesis involves addressing this problem so that the synthesized motions more closely match those seen in nature.

Optimization-based approaches for synthesizing animation motion such as that described in the previous chapter have the advantage that they can solve for a character motion given relatively little information, in particular just a skeletal description of the animal's shape, a description of the laws of physics, and a cost function assigning to each potential motion a notion of the 'effort' required by the animal to perform the motion. Although the animal's skeleton and the laws of physics tend to be relatively straightforward to specify, the notion of the 'effort' required to perform a motion is rather vaguely defined. This vagueness reflects a real difficulty in determining a precise ranking in which natural-looking motions are preferred over unnatural ones.

There are, in fact, many potential ways in which a notion of the effort required for an animal to perform a motion might be defined, and different definitions of this effort will generally result in different optimal motions for the animal. In this optimization-based view taken to the motion synthesis problem in this proposal, the primary culprit for this lack of realism then lies in an incorrect or incomplete specification of the cost function the animal's 'natural' motion should minimize.

The standard approach for solving this problem in the existing literature is to hand-specify this cost function based on biomechanical principles, and in some cases intuition and

experimentation. An example of this approach was taken in the previous chapter (chapter 3) where I proposed a cost function based, among other things, on minimizing a term roughly related to the energy required to perform the motion. In keeping with the prior graphics literature this included minimizing the torque required at each of the animal's joints during the motion. Torque minimization by itself often resulted in unsatisfactory motions, however, so additional terms were included to, for instance, minimize head motion and the rotational velocities the joints during the gait.

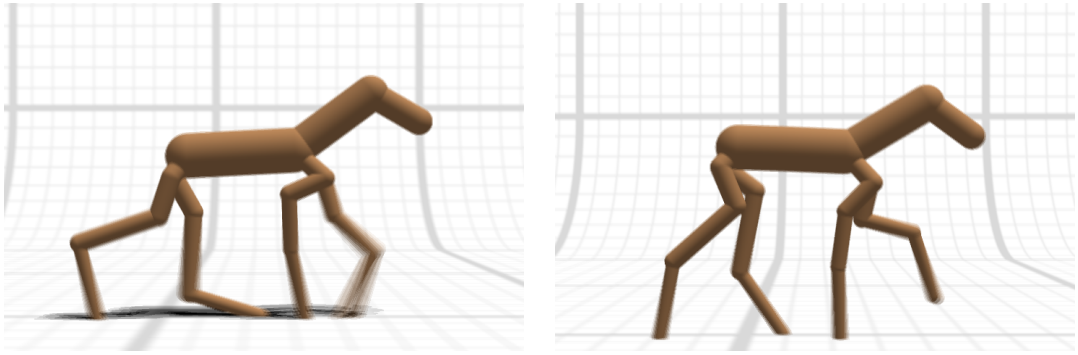


Figure 4.1: The left image shows a frame from a gait for a gazelle as synthesized by the method in chapter 3. The image on the right shows a frame the 'correct' motion of the same gazelle as extracted from a video.

The objective function resulting from this (section 3.2.3) is relatively sophisticated in so far as objective functions for spacetime constraints synthesis methods go, but the results still exhibit relatively substantial differences from real-world motions (figure 4.1). This lack of realism in the synthesized motions should perhaps not be too surprising. In reality the factors underlying an animal's motion are probably quite complex. For instance, although energy minimization likely plays an important role in determining how an animal moves, other factors such as minimizing potentially damaging stresses at joints or along bones, minimizing muscle stress, and coping with uncertainty in the environment almost certainly play a role as well. Further complicating the matter is the fact that the physical properties of real animals and the environments in which they move are substantially more complicated than the mathematical model described in the preceding section.

Rather than attempting to capture this likely extremely complicated objective function

directly, I attempt to automatically approximate it from data of the motions of real-world animals. In an intuitive sense, it might be expected that the 'correct' cost function for a particular animal should resemble the cost functions for other animals with a similar shape. Continuing with the reasoning, my method for improving the realism of the synthesized gaits automatically estimates a cost function for an input animal by interpolating from a database of 'ground truth' cost functions for other animals reconstructed from video data of animal motions. This technique is not only effective in improving the realism of the motions over the approach given in chapter 3, but also performs better than approaches which directly interpolate the motions of animals in the motion database (instead of interpolating their associated cost functions). Furthermore, my approach allows the easy synthesis of motions for animals such as dinosaurs with shapes substantially different than any living creature.

4.1 Background

There are two extremes to the types of approaches which might hypothetically be taken to synthesize more realistic motions. At one end of the spectrum one might attempt to increase the biomechanical fidelity of the model underlying the gait synthesis process. Such an endeavor would likely include modeling the individual muscles in an animal as well as a more accurate model of the actuation properties of muscles and the characteristics of tendons a ligaments. Unfortunately, specifying such a biomechanical model for a new animal can be quite labor intensive (see [44] for an example of modeling the human upper body to a high level of detail). Furthermore, for extinct and fictional animals there may be incomplete or no information with a high level of detail about the animal's biomechanical structure, and there is a similar difficulty in modeling the physical properties environment in which an animal moves.

Nevertheless, when restricted to humanoid bipedal locomotion this style of approach has yielded some success. Recent instances of approaches in this direction include a more accurate biomechanical and metabolic modeling of leg muscles [81] and the modeling of soft tissue deformations in the feet [34]. The first of these is perhaps the most relevant, as the inclusion of accurate muscle models is shown to lead to a quantitatively better match between the synthesized motions and actual human motion capture data than is achieved

by a synthesis method designed without such a muscle model.

There is, however, reason for some skepticism about the potential success of these approaches when applied to the automatic synthesis of motions in animals with highly diverse shapes. Firstly, as a practical consideration, current instances of these approaches rely on synthesis approaches which cannot yet be applied uniformly to widely varying types of animals. Secondly, there is the more theoretical concern that it is not at all clear that a complete model of an animal's physical makeup is even theoretically sufficient to synthesize accurate motions. Instead one might expect that some properties of an animal's motion arise for reasons beyond its pure biomechanical structure. For instance one might hypothesize that some animals expend a little extra energy in order to keep their heads still so that they might visually perceive their environment more accurately. There are numerous other conjectures about the factors underlying realistic animal motion that one might make, and determining which factors are important and to what degrees is not specified directly by the animal's mechanics.

On the other end of the spectrum one might attempt to synthesize an animal's gait by forgoing any consideration of the principles underlying the motion and simply attempting to match the appearance of the motions for animals for which data exists. This approach has its own advantages. Firstly, it can potentially be very simple and efficient to compute a motion for a new animal, since it is unnecessary to solve a complex optimization problem, but rather only to interpolate motions for similar animals. Secondly, if a new animal happens to be very similar in shape to an animal for which there is data, then this approach would be expected to give a very accurate prediction of the motion for the new animal.

Unfortunately, one major difficulty of this approach based on direct data interpolation is that it is not clear how well it can work when applied to an animal that has a shape or mass substantially different than anything data is available for. Indeed, the primary successful application of this approach to creatures with diverse shapes has thus far been limited to highly stylized motions [27]. Unfortunately these cases where the animal is relatively unlike anything one could obtain data for cover some of the most interesting applications of computation locomotion synthesis. A restriction to synthesizing gaits only for animals similar to those that exist would preclude the generation of gaits for most dinosaurs and

many other extinct animals.

Given that each of these two extremes on the spectrum of approaches to the problem of accurate synthesis are subject to relatively severe problems, my method takes a middle ground and combines data-based interpolation with biologically- and physically-motivated factors. An instance of this type of 'middle ground' approach has been previously applied to human locomotion [48, 43] by learning the passive actuation characteristics at a character's joints from a sequence of motion capture and then using this information to create new motions for the same character. Although these passive actuation characteristics are relatively simple biologically-motivated entities approximating the nature of tendons and ligaments, their particular values were derived from a sequence of motion capture data. My method applies this type of approach to the synthesis of animals with widely varying shapes. The primary complication that arises in doing so is that it is allowed that motion will be synthesized for an animal for which there is no motion capture data available.

4.2 Algorithm Overview

At a high level, my approach begins with a database of the shape of different animals $A_1, \ldots, A_n$, each defined by a kinematic skeleton of limbs connected by joints as well as associated information about the mass of each limb, location of the feet and head, etc. as in chapter 3. For each animal A_i an associated motion M_i is given describing how that animal actually moves in nature. For each animal and its associated motion (A_i, M_i) in the database the values for a small number of parameters relevant to the animal's locomotion are estimated such that these parameters, along with A_i are sufficient to synthesize M_i. Given a new animal, the values of these parameters can then be interpolated from the values estimated for similar animals in the database. This then allows a gait to be synthesized for the new animal which hopefully matches the style exhibited by similar animals in the database.

More specifically, this approached is based around the fact that in optimization-based approaches to motion synthesis there is some flexibility (or equivalently, some uncertainty) in defining what specifically an animal's motion should be optimal with respect to. Clearly *some* concept of mechanical efficiency should be included in this optimality criterion in

order to avoid exorbitantly energetically expensive motions, but given this there still remain a number of competing factors to be weighed in determining what an 'optimal' gait should be.

To formalize this notion of flexibility in the definition of 'optimal' used to synthesize a gait, the objective function in the spacetime constraints formulation given in section 3.2.3 is modified to take as input an additional vector ϕ of *inverse parameters*. These inverse parameters allow the definition of what 'optimal' means for a specific animal to be modified, for instance weighting torque minimization more or less heavily with respect to head stability. Different assignments of values to these inverse parameters will therefore in general lead to different gaits being generated by an associated spacetime constraints optimization.

The inverse parameters allow modifications to the definition of optimality, and thereby the style of an optimal gait. Given a database of animal motions, my approach then attempts to estimate values for these inverse parameters which might give rise to each motion in the database. Unfortunately it is not immediately obvious what particular values should be assigned to the inverse parameters in order to achieve a given style in the resulting gait. Solving for the inverse parameters necessary to reproduce each of the gaits in the motion database involves solving an *inverse optimization* problem for each motion.

The output of these inverse optimizations consists of a series of tuples (A_i, M_i, ϕ_i) such that if a spacetime constraints optimization is performed on the animal A_i with the objective function defined by the inverse parameters ϕ_i, then the motion resulting from this optimization will closely resemble M_i. In essence ϕ_i can be roughly thought of as representing an estimate of what the animal A_i cared about doing well in its gait, and which things were less important to perform well.

Once each animal A_i is associated with a set of ground truth values for the inverse parameters ϕ_i, it remains to generate a motion for a new animal A_{new}. Intuitively this is done by generating new values of the inverse parameters ϕ_{new} by interpolating from each ϕ_i weighted according to the similarity between A_i and A_{new}. At first this may seem to be a basic regression problem, but in fact simply applying standard regression techniques does not work well in this case.

The problem with using basic regression on the (A_i, ϕ_i) pairs is that in practice the values of ϕ_i do not necessarily form a coherent pattern. This should not really come as much a surprise, as the value of each ϕ_i were solved for completely independently of the other parameters. To remedy this, a final inverse optimization is performed which attempts to jointly modify all of the ϕ_i parameters for every animal such that the new values *do* allow easy regression when considered together while simultaneously still individually giving rise to motions which closely match each associated M_i.

Once this final inverse optimization is performed, basic regression techniques can then be applied to determine the values of the inverse parameters ϕ_{new} should be used for a new animal A_{new}. These ϕ_{new} allow a gait to be automatically synthesized for A_{new} with a spacetime constraints optimization.

In the remainder of this chapter I will start by describing how the form of the A_i animals and the M_i motions in the motion database are extracted from video footage of actual animal gaits. With a motion database created using this procedure, I will then describe the particular set of inverse parameters ϕ_i used to model the cost function for each animal, and how the values of these inverse parameters are solved for. After this I will describe the joint inverse optimization which adjusts the ϕ_i inverse parameters so that they become better suited to regression techniques. Finally, I will conclude by showing some results from this synthesis method on both living and extinct animals.

4.3 Data Acquistion

As my approach to motion synthesis involves (indirectly) interpolating from a database of ground-truth motions, the first step in the synthesis process is to acquire such a database in the first place. This poses somewhat of a problem, since although motion capture is readily available for humans and a few other animals such as horses and dogs, there is not sufficient motion capture data available to build a motion database covering a wide range of animal shapes.

Fortunately, although there is no database of 3D motion capture spanning a wide range of different animals, a large source potential motion data does exist in the form of raw 2D video footage. The most easily accessible instances of video footage of animal gaits come from

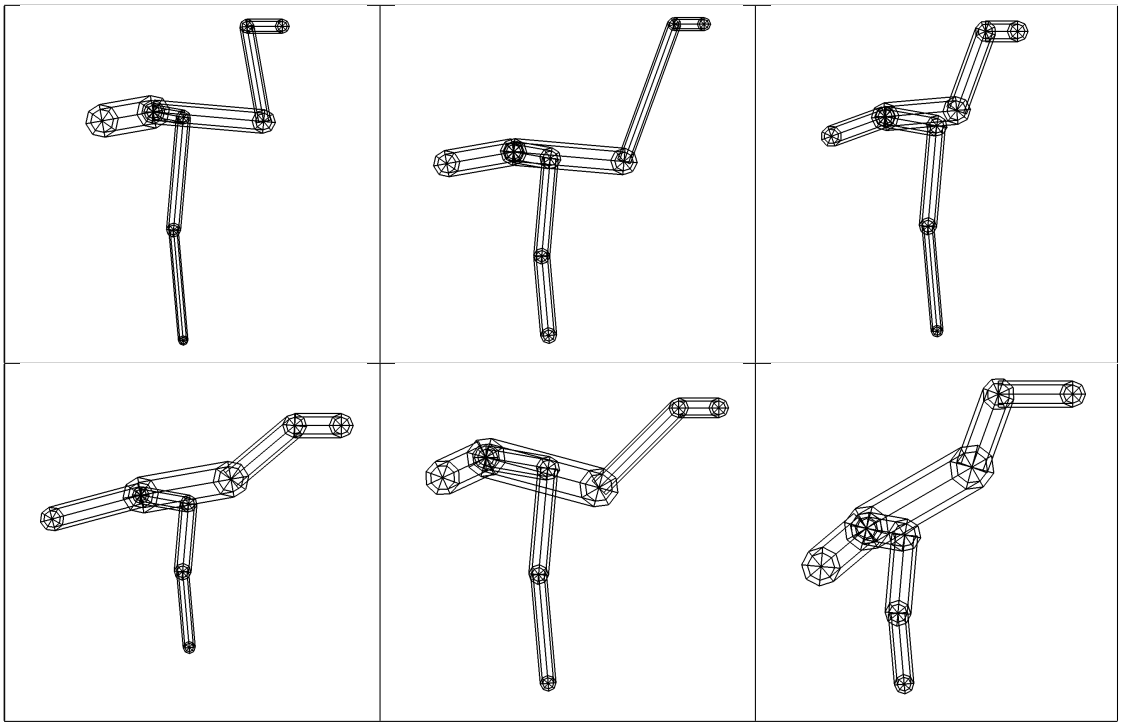


Figure 4.2: The six bipeds in the motion database. From top left to bottom right, greater flamingo, ostrich, secretary bird, greater roadrunner, emu, and southern ground hornbill.

video sharing sites such as YouTube and Flickr Video. The videos available through these and other sites are of sufficient variety to provide examples of the gaits of many different animals. The motion database used to generate the examples in this thesis consists of walking motions for a range of both bipedal and quadrupedal animals. Information about the size and mass of the bipedal animals included in the motion database is given in figure 4.3 while an illustration of the kinematic skeletons for the bipedal animals is shown in figure 4.2. The respective information for the quadrupeds included in the motion database is given in figures 4.5 and 4.4.

Although in principle online video sharing sites provide a sufficient database of animal motion, it still remains to actually convert the motion implicit in each video into a usable form. The motions implicit in each video will eventually be used as ground truths for 3D gait synthesis, so a cyclic 3D gait cycle must be extracted from each video. Doing this completely automatically remains an unsolved computer vision problem, so the extraction

animal	height	mass
greater flamingo	1.1m	3kg
ostrich	2.3m	94.5kg
secretary bird	0.8m	3.3kg
greater roadrunner	0.22m	0.3kg
emu	1.5m	30.0kg
southern ground hornbill	0.9m	3.6kg

Figure 4.3: The bipeds included in the motion database along with their associated heights and masses.

of a 3D gait from each video is instead accomplished in a semi-automatic manner. In order to simplify this process, the videos are assumed to be shot side-on from a camera with minimal non-rotational motion.

There are several steps involved in adding a new animal and its associated motion to the motion database:

1. Determine the length of each of the animal's limbs

2. Determine the mass of each of the animal's limbs

3. Track the background in the video

4. Track a set of points on the animal

5. Extract gait cycle from tracked points.

6. Create a physically realistic 3D motion matching the extracted gait cycle.

The first two of these steps relate to creating a model of the animal which corresponds reasonably accurately to that in the video. This is done by selecting a single frame from the video to use as a reference. From this frame the relative lengths of the limbs of the animal are measured, and absolute values for these lengths are determined by uniform scaling so that the animal matches a pre-specified height (shown in figures 4.3 and 4.5).

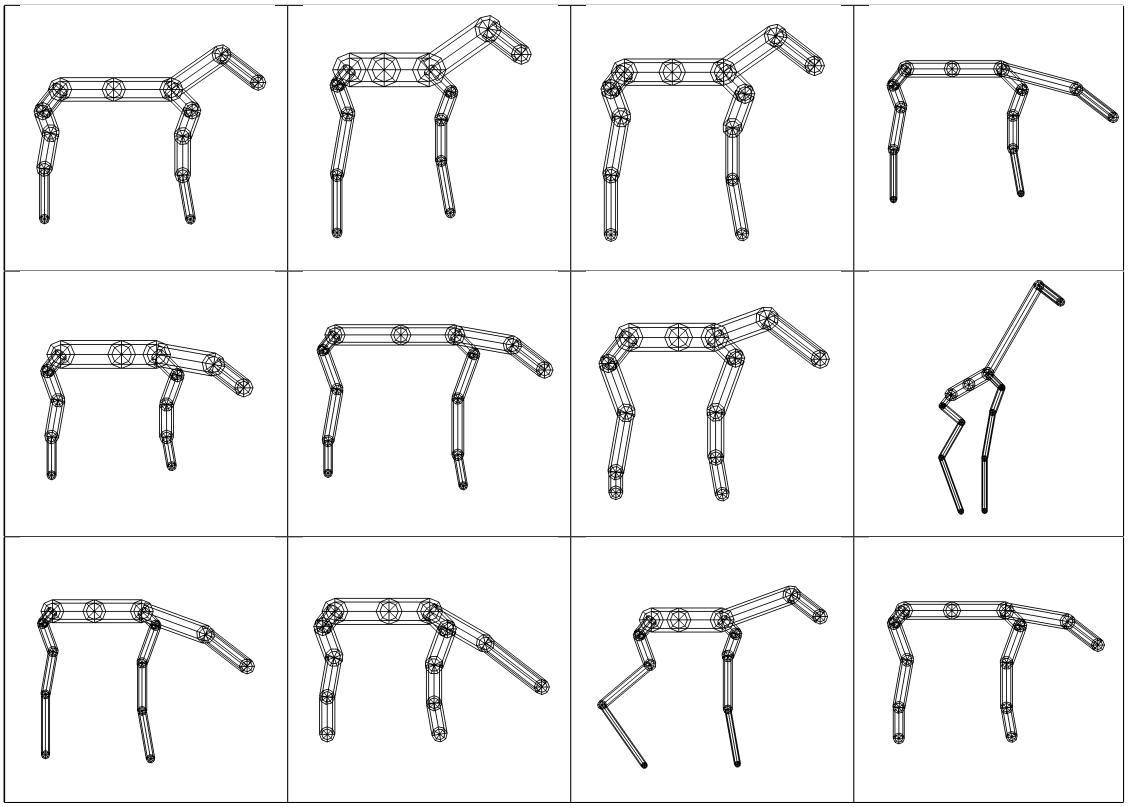


Figure 4.4: The twelve quadrupeds in the motion database. From top left to bottom right, horse, Thomson's gazelle, pronghorn antelope, bison, house cat, cheetah, elephant, giraffe, moose, rhinoceros, steenbok, tiger.

Once the length of each of the animal's limbs has been determined, the masses of these limbs are again estimated based on the reference image in a similar manner by first determining their relative masses based on their volume and then choosing the absolute masses so that the total mass of the animal is as given in figures 4.3 and 4.5. The limbs constituting the legs of the animal are approximated by cylinders and their relative masses determined accordingly. The masses of the torso, neck, and head are determined by approximating the body of the animal as a generalized cylinder along with hand-specified weights defining what proportion of the mass of each segment in the cylinder should be attributed to each of the limbs nearby (figure 4.6). This same process is also used to model animals for which video data is not available.

animal	height	mass
horse	1.55m	500kg
Thomson's gazelle	0.55m	20kg
pronghorn antelope	0.9m	50kg
bison	1.7m	700kg
house cat	0.24m	4.5kg
cheetah	0.85m	50kg
elephant	3.7m	5200kg
giraffe	5.5m	1000kg
moose	1.8m	320kg
rhinoceros	1.6m	1600kg
steenbok	0.5m	17kg
tiger	1.0m	180kg

Figure 4.5: The quadrupeds included in the motion database along with their associated heights and masses.

In addition to building a model of each animal, a gait for the animal must be extracted from each video. The first step in doing this is to determine the motion of the video camera, which is achieved by tracking SURF features between the frames in the video followed by solving for the optimal global motion of the camera based on these features. Once the camera motion is determined, it is subtracted from the video so that any remaining motion is due to the movement of the animal and not that of the camera.

Next, a number of points on the animal are manually tracked across the video. These points vary somewhat from animal to animal, but always include at least two points on the torso, at least on on the head, as well as points on the knees and feet. This gives a series of 2D traces for each of a set of points on the animal traced through each frame in the video. A sub-segment of the frames in the video is then selected for which these traces which represent a single gait cycle. In addition, a few frames are chosen and the 3D model for the animal is manually adjusted to be superimposed over the animal in these frames in the video. Once each point has been paired with an associated limb, this allows the coordinates of each point to be determined in a coordinate system local to the point's associated limb, and thereby allows the accuracy with which a 3D gait matches the traces of the points in the video to be determined.

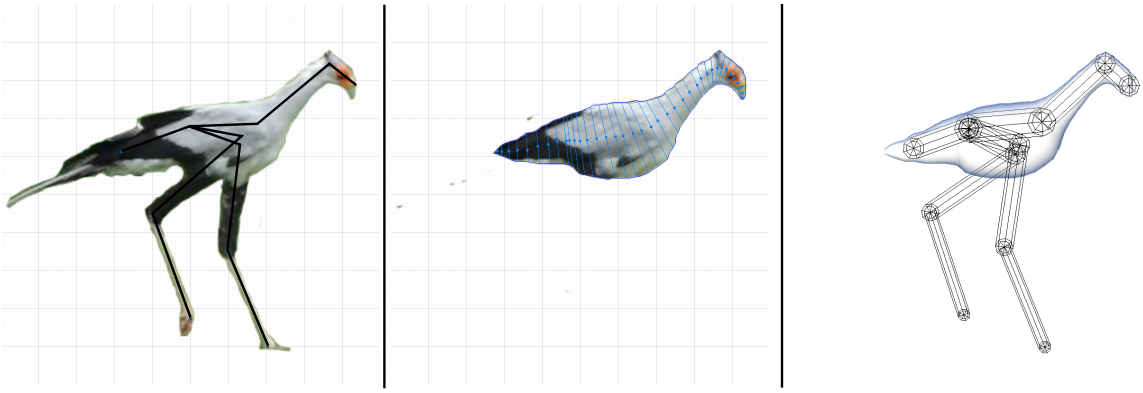


Figure 4.6: A basic outline of the steps involved in creating a new animal. First the lengths of the skeleton's bones are measured using an image as a reference. Next the shape of the skeleton's torso is traced and modeled with a generalized cylinder. The masses of the torso, head, and neck limbs are computed from this generalized cylinder, while the masses of the leg limbs are computed assuming the limbs are cylindrical.

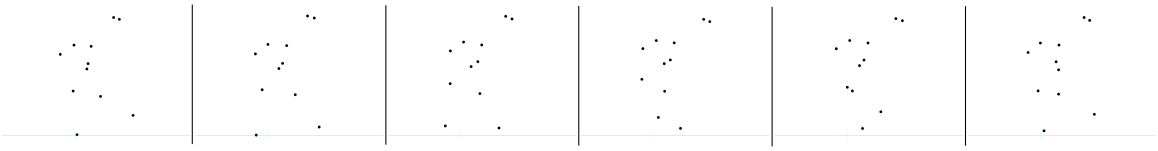


Figure 4.7: An example of the points extracted from a gait for a secretary bird (only a subset of the frames are shown).

Once these steps have been completed a series of 2D point positions representing a single gait cycle for the animal have been extracted from the video (figure 4.7). The final step in the data acquisition process involves fitting a physically realistic 3D motion for the animal to these points traces. For each frame i of the video let $d_{i,j}$ denote the 2D position of the jth point in the traces extracted from data, and denote and the 2D X-Y projection associated point on the animal in a motion M with $p_j(M(f_i))$.

If M were a perfect fit to the data then it would hold that $d_{i,j} = p_j(M(f_i))$ for each frame i and point j. Since in general a perfect fit to the data will not be possible, the degree to which a motion M matches the data is quantified with a sum of squared distances between each point in the data and the associated point on the animal in the motion M:

$$\sum_i \sum_j w_j \| d_j - p_j(M(f_i)) \|_2^2 \tag{4.1}$$

Here w_j is a weight assigned to each point on the animal to give preference to a more accurate fit of the knee and foot positions (weight of 3) while allowing greater errors in the fit of points on the torso (weight of 0.5), all other points having weight 1.0.

Equation 4.1 is smooth and differentiable, so it is suitable for inclusion in a spacetime constraints optimization such as that described in equation 3.1. This allows a 3D gait closely matching the data to be generated as the solution of the nonlinear programming problem:

$$\begin{aligned} M^* = \underset{M}{\operatorname{argmin}} \frac{1}{n} \sum_i \left[\sum_j w_j \| d_j - p_j(M(f_i)) \|_2^2 + \alpha \cdot cost(M(f_i)) \right] & \\ \text{s.t. } g_j(M(f_i)) = 0 & \quad \forall j, i \\ h_k(M(f_i)) \leq 0 & \quad \forall k, i \end{aligned} \tag{4.2}$$

where $cost(M(f_i))$, $g_j(M(f_i))$, and $h_k(M(f_i))$ are all defined as in equation 3.1. Some contribution of $cost(M(f_i))$ is still included in the objective as a regularization term and helps to avoid overfitting to the data. The term α controls how strongly minimizing $cost(M(f_i))$ is weighted versus minimizing the data fit error. Larger values of alpha tend to lead to smoother motions at the cost of a poorer data fit. Although α is occasionally altered on a per-animal basis, a value of 10^{-5} works well for most of the animals in the database, with other values ranging from 10^{-3} to 10^{-8}.

4.4 Inverse Optimization

A completed database of animal skeletons $A_1, \ldots, A_m$ and their associated ground truth motions $M_1, \ldots, M_m$ describes how each animal should move in a strictly kinematic sense, but it gives little direct information about how an animal not found in this database should move. Such a motion database does, however, give a good deal of indirect information about the motion of a new animal. In particular, it is likely that the 'style' of the motion for a new animal will be similar to the styles of the motions of the animals in the motion

database with a form similar to the new animal. While it is tempting to make the leap that this further implies that the motions themselves should be similar, it turns out that considering the motions directly doesn't allow easy generalization to new animals with a form moderately different than anything in the database. It is instead better to represent the style of a motion is a way which does not directly reference the motion itself.

An ideal method for representing the style of motion exhibited by a specific animal would be both compact, easily generalizable, and biologically meaningful. Although the motion database itself does immediately provide any such representation, one might still hope to indirectly infer a suitable representation from it. This approach requires two components: a representation which is is capable of approximately representing each gait in the motion database and a way to learn the parameters of the representation required to reconstruct a specific motion.

Within an optimization based motion synthesis framework the most fundamental way of altering the style of a synthesized motion is to alter the specifics of the definition of optimality under which the motion is synthesized. This type of 'tweaking' the spacetime constraints objective function is already present in a relatively crude form within existing optimizers. For instance, in the synthesis method described previously in chapter 3 there are four weights (C_1, C_2, C_3, C_4 in equations 3.14, 3.15, 3.16, and 3.17) governing how much preference is given to minimizing joint torques, joint velocities, translational motion of the head, and rotation of the head.

Each different setting of these four weights leads to a different notion of what counts as an optimal gait, and thereby results in differing styles in the synthesized gaits. In chapter 3 these four weights were specified by hand and taken as fixed. In principle, however, these weights could be automatically adjusted in order to achieve some desired style in the resulting gait. Of course, altering the four particular weights used in equations 3.14, 3.15, 3.16, and 3.17 does not provide sufficient variability in the resulting gaits to accurately represent the variation in the gaits within the motion database. The larger and more expressive set of parameters described in this section, however, is sufficient relatively accurately reproduce all of the motions in the motion database.

In order to allow the spacetime constraints objective function to be modified so that the

style of the resulting synthesized motions can be controlled, it is necessary to describe the space of possible ways in which this function may be altered. To this end a vector of real-values parameters termed the *inverse parameters* is specified. A particular value for this vector, denoted by ϕ, describes an associated cost function suitable for use in a spacetime constraints optimization such as those in chapter 3. Before presenting the specific choice of the inverse parameters in ϕ used to create the examples given in this chapter I will first describe how the inverse parameters are used and learned in general, deferring a description of the specifics of these parameters to section 4.4.1.

In this context, the objective function used in an optimization is defined as the average per-frame cost, where the cost function at each frame, $cost(M(f_i), \phi)$, now depends not only on the motion of the animal at that frame in the animation $M(f_i)$, but also on the values of the inverse parameters ϕ. Within a single spacetime constraints optimization, ϕ will remain fixed and only M will be solved for by the optimizer in light of the choice of ϕ.

Since ϕ allows the style of a synthesized motion to be easily altered, the next step is to automatically solve for ϕ so that the resulting motion closely matches that animal's 'true' motion from the motion database. This process is known as *inverse optimization*, so named because rather than starting with an objective function and producing a motion as in traditional optimization, it proceeds in the reverse order and from a 'ground truth' motion attempts to determine an objective function for which the given motion is optimal. This paradigm has been successfully applied to the synthesis of stylized human locomotion by inferring spring and damper values describing the passive actuation at each of a character's joints [48].

The goal of inverse optimization is, given and animal A_i and its associated ground truth motion M_i, to determine the vector of inverse parameters ϕ_i such that the motion resulting from a spacetime constraints optimization with cost function $cost(\cdot, \phi)$ is as close as possible to M_i. Formalizing this process requires a way of measuring the similarity between a pair of motions. This is done with a distance metric $D(M_a, M_b)$. Although the most obvious definition of this distance metric would be sum-of-squared distances between the joint angles in M_a and M_b, the visual quality of the results as well as the time required to solve of ϕ_i can be improved with a slightly more involved definition where $D(M_a, M_b)$ is written as the

sum of five terms, each computed as a sum over all the frames in the motions of the squared difference between: the degrees of freedom of the poses (4.3), their velocities (4.4), their accelerations (4.5), the height of the feet (4.6), and the acceleration of the head (4.7).

$$
\begin{aligned}
D(M_a, M_b) = \sum_i \Bigg[& \\
& \sum_j w_j C_1 \left(\mathbf{q}_b(i,j) - \mathbf{q}_a(i,j)\right)^2 + && (4.3)\\
& \sum_j w_j C_2 \left(\dot{\mathbf{q}}_b(i,j) - \dot{\mathbf{q}}_a(i,j)\right)^2 + && (4.4)\\
& \sum_j w_j C_3 \left(\ddot{\mathbf{q}}_b(i,j) - \ddot{\mathbf{q}}_a(i,j)\right)^2 + && (4.5)\\
& \sum_j C_4 (p_b(i,\text{foot}_j)_y - p_a(i,\text{foot}_j)_y)^2 + && (4.6)\\
& C_5 \left(\left(\ddot{p}_b(i,\text{head}_1) - \ddot{p}_a(i,\text{head}_1)\right)^2 + \left(\ddot{p}_b(i,\text{head}_2) - \ddot{p}_a(i,\text{head}_2)\right)^2 \right) \Bigg] && (4.7)
\end{aligned}
$$

where $\mathbf{q}_a(i,j)$ and $\mathbf{q}_b(i,j)$ give the values of the jth degree of freedom in frame i within motions M_a and M_b respectively, $\dot{\mathbf{q}}_{\{a,b\}}(i,j)$ and $\ddot{\mathbf{q}}_{\{a,b\}}(i,j)$ give the time derivatives of $\mathbf{q}_{\{a,b\}}(i,j)$ (computed by finite differences), $p_{\{a,b\}}(i,\text{foot}_j)$ gives the 3D position of the jth foot in frame i, and $\ddot{p}_b(i,\text{head}_1)$ and $\ddot{p}_b(i,\text{head}_2)$ give the accelerations of a point in frame i on the tip or base of the head respectively. In my implementation I use $C_1 = 0.25$, $C_2 = 0.6$, $C_3 = 0.5$, $C_4 = 1.5$, $C_5 = 1.8$.

Given this definition for $D(M_a, M_b)$, an inverse optimization problem can be described as optimizing for the values of ϕ_i which yield a minimal value of $D(\text{opt}(A_i, \phi), M_i)$:

$$\phi_i = \underset{\phi}{\text{argmin}}\, D(\text{opt}(A_i, \phi), M_i) \tag{4.8}$$

where $\text{opt}(A_i, \phi)$ is the motion resulting from a spacetime constraints function using the animal A_i and the cost function defined by ϕ:

$$\begin{aligned} \text{opt}(A_i, \phi) = \underset{M}{\text{argmin}} \; & \frac{1}{n} \sum_i cost(M(f_i), \phi_i) \\ \text{s.t. } & g_j(M(f_i), \phi) = 0 \qquad \forall j, i \\ & h_k(M(f_i), \phi) \leq 0 \qquad \forall k, i \end{aligned} \tag{4.9}$$

Intuitively, an inverse optimization finds the vector of inverse parameters ϕ_i such that the motion synthesized for animal A_i most closely matches the motion M_i associated with A_i in the motion database.

Although there is at least one existing algorithm which attempt to solve this sort of problem relatively efficiently [48], it can give poor results in the case that the inverse parameters have substantially different scalings. To avoid this difficulty, equation 4.8 is solved directly using a standard covariance matrix adaptation (CMA) optimization [24] as an outer loop. The CMA optimization samples different values for ϕ_i and for each evaluates $D(\text{opt}(A_i, \phi), M_i)$. This involves solving the spacetime constraints problem given in equation 4.9 for each sampled ϕ_i, making the resulting optimization relatively slow. To alleviate this my implementation parallelizes the evaluations equation 4.9 and runs the optimization on a cluster of computers.

4.4.1 Inverse Parameter Definitions

A critical component to employing inverse optimization in the synthesis of animal motions is the choice of inverse parameters represented by ϕ. In previous work [48] these inverse parameters have been chosen to represent the passive actuation characteristics at each of a character's joints. For the synthesis of gaits for diverse animals, however, this proves to be insufficient and further inverse parameters are needed. A table summarizing the full set of inverse parameters is given in table 4.1, and I will next describe how these parameters are incorporated into the optimization's cost function.

Ground Height Uncertainty Parameters

The inverse parameters h_g, w_g, and w_{h_g} are used to approximate the fact that in the real world an animal would not be absolutely certain about the location of the ground, and would thus alter its gait to reflect this uncertainty. Modeling uncertainly in the environment of this sort has been done in the context of dynamic controllers [80], but incorporating this uncertainly into a spacetime constraints formulation is more difficult, particularly as the use of the spacetime constraints solver in the inner loop of the inverse optimization requires that the solver be both (relatively) efficient and robust.

The penalties related to the height of the foot are approximated by introducing a penalty based on the sum of two terms related to the horizontal and vertical motion of the foot. In the following equations y will represent the height of the foot at some given frame i and y_2 will represent the height of the foot in frame $i+1$. In addition $\mathbf{v}_{xz}$ and v_y will represent the horizontal and vertical components of the velocity of a given foot relative to the ground.

Although, strictly speaking, in the spacetime constraints optimization the ground is always at height $y_{\text{ground}} = 0$, a penalty is nevertheless introduced modeled on the assumption that a contact with the ground might still occur even if the foot is higher. In particular the 'actual height' of the ground is assumed to be distributed with a prior probability given by the exponential distribution $h_g e^{-h_g y}$.

For the vertical component of the ground-contact penalty, note that since the foot is at height y in frame i, the ground must be at height at least y. Conditioned on this restriction and given the aforementioned prior of the ground height as $h_g e^{-h_g y}$, the posterior distribution of the ground being at height y_2 is 0 if $y_2 >= y$ (or equivalent $v_y > 0$) and if $y_2 < y$ is given by:

$$\frac{h_g e^{-h_g y_2}}{\int_0^y h_g e^{-h_g y_2} dy_2} = \frac{h_g e^{-h_g y_2}}{1 - e^{-h_g y}} \tag{4.10}$$

The probability of the ground being between y_2 and y, and thus the probability of an 'unexpected contact' due to the vertical motion of the foot between frame i and $i+1$ is then:

$$p_{c_y} = \int_{y_2}^{y} \frac{h_g e^{-h_g y_2}}{1 - e^{-h_g y}} dy_2 = \frac{e^{-h_g y_2} - e^{-h_g y}}{1 - e^{-h_g y}} \tag{4.11}$$

For the horizontal motion of the foot the probability of a contact is simply taken to be the probability that the ground is at height at least y multiplied by a constant factor of w_{h_g} approximating the 'bumpiness' of the ground, giving $p_{c_x} = w_{h_g}(1 - e^{-h_g y})$. A penalty is then added to the objective function computed by a weighted combination of the probability of a contact multiplied by the velocity of the foot in the horizontal and vertical directions:

$$(e^{w_g} - 1)\left(p_{c_x}\|\mathbf{v}_{xz}\|_2 + \max\left\{0, -p_{c_y}\mathbf{v}_y\right\}\right) \tag{4.12}$$

Near-Ground Resistive Forces

In addition to adding a penalty based on an approximation to the probability of an unexpected contact with the ground, inverse parameters are also introduced approximating near-ground resistive forces resulting from shrubs, undergrowth, shallow water, etc. To this end the parameters h_w and d_w are used to add forces to each foot in a direction opposite its motion when the foot is near the ground.

These resistive forces are computed based on a simple fluid drag equation which assumes an immersed cross-section area for each foot proportional to $h_w - y$ and a drag coefficient of e^{d_w}. The force at each foot is then given by:

$$\mathbf{f} = -\mathbf{v}\|\mathbf{v}\|_2 e^{d_w} \max\{0, h_w - y\} \tag{4.13}$$

Although these forces do not directly add penalties to the objective function, they do alter it indirectly by requiring larger torques in order to move the foot quickly when near the ground.

Leg Coactuation

The standard spacetime constraints formulation used to synthesize motions treats each joint in the animal as being capable of moving entirely independently of all the other joints. In

reality, however, some pairs of joints exhibit a tendency to be *coactuated* such that their motions tend occur in concert rather than independently. Rather than directly modeling the muscles responsible for this as in [81], a simplified approach is taken where a penalty is added to the objective function based on the relative velocities of selected pairs of joints.

In particular, this coactuation is modeled between the knee and ankle joints in an animal's legs. For a biped the inverse parameters c_l and c_{r_l} control a penalty is added to the cost function based of the degree to which the relative rotational velocities of the knee and ankle differ from the ratio given by c_{r_l}:

$$(e^{c_l} - 1)(c_{r_l}\dot{\mathbf{q}}_{\text{knee}} - \dot{\mathbf{q}}_{\text{ankle}})^2 \tag{4.14}$$

Where $\dot{\mathbf{q}}_{\text{knee}}$ and $\dot{\mathbf{q}}_{\text{ankle}}$ are the rotational velocities at the knee and ankle respectively. For a quadruped the inverse parameters c_a and c_{r_a} are included to add an additional penalty related to the relative velocities of the elbow and wrist joints.

Joint-Torque Weights

In the spacetime constraints optimization described in chapter 3 a penalty was added to the objective function based on the sum of the squared magnitudes of the torques required at each joint in order to perform a motion (equation 3.14). In reality, however, some joints are stronger than others and the torques should not be penalized uniformly. The inverse parameters t_h, t_k, and t_a are thus used as weights multiplying the costs corresponding to the squared magnitudes of the torques at the hip, knee, and ankle joints respectively. For quadrupeds the additional inverse parameters t_s, t_e, and t_w are added to scale the costs at the shoulder, elbow, and wrist joints.

Other Inverse Parameters

The cost function given in section 3.2.3 involved contributions from several different terms, each weighted by a pre-specified constant. In the inverse optimization, these weights are solved for automatically rather than manually fixed beforehand. In particular the inverse parameters w_f, w_a, and w_h are included to weight the contribution of costs arising to forces

transmitted through the animal's joints, the angular acceleration of the animal's joints, and the amount of motion in the head respectively.

In addition, in keeping with the approach previously taken for the inverse optimization of human motions in [48] inverse parameters are added to control the passive actuation characteristics at a subset of the animal's joints. In the optimization previously described, section 3.2.2 specifies how spring, spring rest angle, and damper coefficients for each joint can be automatically optimized over. In the inverse optimization, however, the spring, spring rest, and damper coefficients for the hip, knee, and ankle are explicitly specified by values of the inverse parameters. For quadrupedal animals the passive actuation of the shoulder, elbow, and wrist joints is also controlled by a set of inverse parameters. A full list of these inverse parameters is given in table 4.1

parameter	biped	quadruped	description
h_g	✓	✓	falloff of ground uncertainty distribution
w_g	✓	✓	weight for vertical ground uncertainty penalties
w_{h_g}	✓	✓	weight for horizontal ground uncertainty penalties
h_w	✓	✓	maximum height of near-ground drag
d_w	✓	✓	drag coefficient for near-ground drag
c_l	✓	✓	weight on knee-ankle coactuation
c_{r_l}	✓	✓	target ratio between knee and ankle velocities
c_a		✓	weight on elbow-wrist coactuation
c_{r_a}		✓	target ratio between elbow and wrist velocities
t_h	✓	✓	scaling for torques exerted at the hip
t_k	✓	✓	scaling for torques exerted at the knee
t_a	✓	✓	scaling for torques exerted at the ankle
t_s		✓	scaling for torques exerted at the shoulder
t_e		✓	scaling for torques exerted at the elbow
t_w		✓	scaling for torques exerted at the wrist
w_f	✓	✓	scaling for joint-force penalties
w_a	✓	✓	scaling for joint-acceleration penalties
w_h	✓	✓	scaling for head stability penalties
k_{s_h}	✓	✓	spring constant for hip
$\bar{q}_h$	✓	✓	spring rest angle for hip
k_{d_h}	✓	✓	damper coefficient for hip
k_{s_k}	✓	✓	spring constant for knee
$\bar{q}_k$	✓	✓	spring rest angle for knee
k_{d_k}	✓	✓	damper coefficient for knee
k_{s_a}	✓	✓	spring constant for ankle
$\bar{q}_a$	✓	✓	spring rest angle for ankle
k_{d_a}	✓	✓	damper coefficient for ankle
k_{s_s}		✓	spring constant for shoulder
$\bar{q}_s$		✓	spring rest angle for shoulder
k_{d_s}		✓	damper coefficient for shoulder
k_{s_e}		✓	spring constant for elbow
$\bar{q}_e$		✓	spring rest angle for elbow
k_{d_e}		✓	damper coefficient for elbow
k_{s_w}		✓	spring constant for wrist
$\bar{q}_w$		✓	spring rest angle for wrist
k_{d_w}		✓	damper coefficient for wrist

Table 4.1: A table of the inverse parameters used to specify the style of an animal's gait. Bipedal and quadrupedal animals have slightly different associated sets of parameters.

4.5 Joint Inverse Optimization

The inverse optimization described in section 4.4 allows a compact and intuitively comprehensible description for an animal's gait to be found, and using this a new gait can be synthesized which closely matches the ground truth motion of the animal. The shortcoming of this technique, however, is that in order to work one must already have a ground truth motion for the animal. Although the vector of inverse parameters can also be used to synthesize novel motions for an animal which are not derived directly from the ground truth gait, an ideal technique would also be able synthesize motions for entirely new animals.

Running the inverse optimization procedure on every animal in the motion database yields a set pairings (A_i, ϕ_i) relating an animal's form A_i to a vector of inverse parameters ϕ_i describing the desired cost function over the animal's potential motions. A simple idea is then to treat the problem of synthesizing a gait for a new animal as a traditional regression problem. That is, given a new animal A_{new}, a vector of inverse parameters for this animal ϕ_{new} are constructed by interpolating between the various ϕ_i vectors based on the similarity between A_{new} and each A_i. This interpolated value of ϕ_{new} can then be used in conjunction with a spacetime constraints optimization to synthesize a motion for A_{new}.

Unfortunately this type of direct regression on the ϕ_i inverse parameter vectors does not work very well in practice. The primary reason for this is that the values of the the ϕ_i vectors are often not well suited to regression. This type of regression, and in particular regression on small datasets (such as the motion database), requires that if A_i and A_j are similar, then so are ϕ_i and ϕ_j. Since all of the ϕ_i inverse parameters are solved for completely independently, this is often not the case.

At the heart of this difficulty with direct regression is the fact that each of the inverse parameters is solved for completely independently of the others. Since the only objective of each optimization is to match the ground truth gaits, it should perhaps come as no surprise that the results of these optimizations are relatively ill-suited to the rather different task of allowing regression across the inverse parameters of different animals to work well. To remedy this the set of inverse parameters for all the animals in the motion database needs to be solved for as a complete set, rather than solving for each ϕ_i independently.

I will refer to this type of optimization in which the entire set of inverse parameters is solved for as a whole as a *joint inverse optimization*. In a joint inverse optimization there are two simultaneous goals. Firstly, each of the ϕ_i should minimize or nearly minimize the distance between the associated synthesized motion and M_i as specified by equation 4.8. Secondly, the set of all the ϕ_i taken together should allow for regression which is as accurate as possible.

Central to the notion of a joint inverse optimization is the *regression function* which takes as input an animal A and returns a vector of inverse parameters ϕ. For any particular choice of regression function, let $R(A, \theta)$ denote the vector of inverse parameters estimated for the animal A where θ is a vector of parameters controlling the regression function (for instance if $R(A, \theta)$ is a linear function of the animal's mass, then θ would be a vector of slope and offset coefficients). Although it is not central to the concept on joint inverse optimization, in my implementation I use a nearest-neighbor regressor, so θ is given directly by the concatenation of the ϕ_i inverse parameters derived from the motion database.

Given a choice of regression function parameterized by θ, the most obvious choice for the formulation of a joint inverse optimization problem is to solve for the value of θ which yields a minimal deviation between the resulting gaits for each animal and the animal's associated ground truth motions:

$$\underset{\theta}{\operatorname{argmin}} \sum_i D(\text{opt}(A_i, R(A_i, \theta)), M_i) \tag{4.15}$$

This direct approach to solving the joint inverse optimization problem unfortunately does not work well in practice. The reason for this lies in the computation of the $\text{opt}(A_i, R(A_i, \theta))$ motions. These motions are each generated as the solution of a spacetime constraints optimization (equation 4.9) which is *not* guaranteed to actually successfully converge to a solution. Indeed, even though the spacetime constraints solver described in chapter 3 is relatively robust, it is still prone to occasional failures, and can often be 'forced' to fail by a particularly bad choice of inverse parameters.

The fact that the computation of the $\text{opt}(A_i, R(A_i, \theta))$ motions can fail makes the direct evaluation of equation 4.16 impossible. In principle one might attempt to fix this by

computing equation 4.16 only over the $\text{opt}(A_i, R(A_i, \theta))$ motions for which the spacetime constraints optimization successfully converges and somehow penalizing settings of the regression parameters by how many of the associated spacetime constraints optimizations fail. Correctly computing this penalty on the number of failed optimizations, however, turns out to be at best quite brittle and very difficult to get right.

A more robust and efficient formulation for joint inverse optimization is to allow the inverse parameters used to solve for an animal's motion to differ from the inverse parameters resulting from the regression function $R(A_i, \theta)$. This treats the regression as a soft constraint rather than a hard constraint:

$$\underset{\theta,\phi_1,\ldots,\phi_m}{\text{argmin}} \quad \sum_i \left[\alpha \left(\|\phi_i - R(A_i, \theta)\|_2^2 + r(\theta)\right) + D(\text{opt}(A_i, \phi_i), M_i)\right] \tag{4.16}$$

where $r(\theta)$ is a *regularization function* designed to prevent overfitting. In cases where $R(A_i, \theta)$ is defined such that the number of regression parameters is significantly less than the number of animals in the motion database, then $R(A_i, \theta) = 0$ may be used. In other cases such as with nonparametric regression, however, an appropriate definition of $R(A_i, \theta)$ is necessary. I will describe the particular definition of $r(\theta)$ used in my implementation shortly.

Notice that for large values of α, and assuming that $R(A_i, \theta) = 0$, the $\|\phi_i - R(A_i, \theta)\|_2^2$ term in equation 4.16 dominates the objective function. Ignoring the possibility that $\text{opt}(A_i, \phi_i)$ might fail, each $\phi_i \approx R(A_i, \theta)$ and the solution to equation 4.16 will closely match a solution to equation 4.15. Note that in the case that $R(A_i, \theta) \neq 0$ very large values of α can cause oversmoothing so a smaller value of α which balances regression errors against errors in the gait data fit should be used.

When accounting for the possibility that $\text{opt}(A_i, \phi_i)$ might fail, however, the optimization defined in equation 4.16 still behaves relatively well. This is because the only term which relies on the result of this optimization is $D(\text{opt}(A_i, \phi_i), M_i)$, which can be computed independently for each animal, and matches the objective function in a normal inverse optimization (equation 4.8). Since the CMA-based technique for solving a straightforward inverse optimization problem is resistant to the possibility that $\text{opt}(A_i, \phi_i)$ might fail, it is

easier to develop an optimization technique for solving a joint inverse optimization problem which shares this capability.

In order to solve the joint inverse optimization problem defined by equation 4.16, first note that the optimization is in a form which is partially decoupled. In particular, only the $\|\phi_i - R(A_i, \theta)\|_2^2 + r(\theta)$ regression error term relates the different animals to each other (via θ), and that remaining term $D(\text{opt}(A_i, \phi_i), M_i)$ can be independently evaluated for each ϕ_i. This leads to an optimization technique in which a series of otherwise independent optimizations for each ϕ_i are coupled together by the regression function $R(A_i, \theta)$ and the regularization function $r(\theta)$. This is achieved in a manner reminiscent of coordinate descent by alternating between minimizing θ and minimizing $\phi_1, \ldots, \phi_m$.

The phase of the optimization minimizing $\phi_1, \ldots, \phi_m$ is is based on the covariance matrix adaptation (CMA) algorithm (see [24] or section 3.3.1 for a description of the CMA algorithm). Since if θ is held fixed equation 4.16 can be minimized independently for each ϕ_i, a mean and covariance matrix is maintained independently for each ϕ_i in a manner identical to a series of independent ordinary inverse optimizations.

At each iteration, this gives a set of samples $\phi_{i,1}, \phi_{i,n}$ for each A_i. Treating these samples as the current estimates for each ϕ_i, equation 4.16 is minimized for θ. As only the $\|\phi_i - R(A_i, \theta)\|_2^2 + r(\theta)$ term depends on θ, solving for θ reduces to a straightforward regularized least-squares optimization and can be solved relatively efficiently. This yields a new estimate for θ, and allows the cost associated with each $\phi_{i,j}$ CMA sample to be computed as defined by equation 4.16, after which the mean and covariance associated with each CMA sub-problem are updated.

Solving a joint inverse optimization problem also requires a setting for the parameter α used to weight the terms in the objective function resulting from how closely each ϕ_i matches $R(A_i, \theta)$. In order to have the solution to the optimization approximate equation 4.15 α should be set to a relatively large value. Unfortunately doing so tends to result in very slow convergence of the optimization. To remedy this a continuation is performed where α is started out at a low value and then gradually increased over the course of the optimization. For quadrupeds I set the value of α in iteration i to $\alpha_i = 0.01 + 0.02\, i$ over a total of 50 iterations while for bipeds I use $\alpha_i = 0.001 + 0.004\, i$ over a total of 35 iterations.

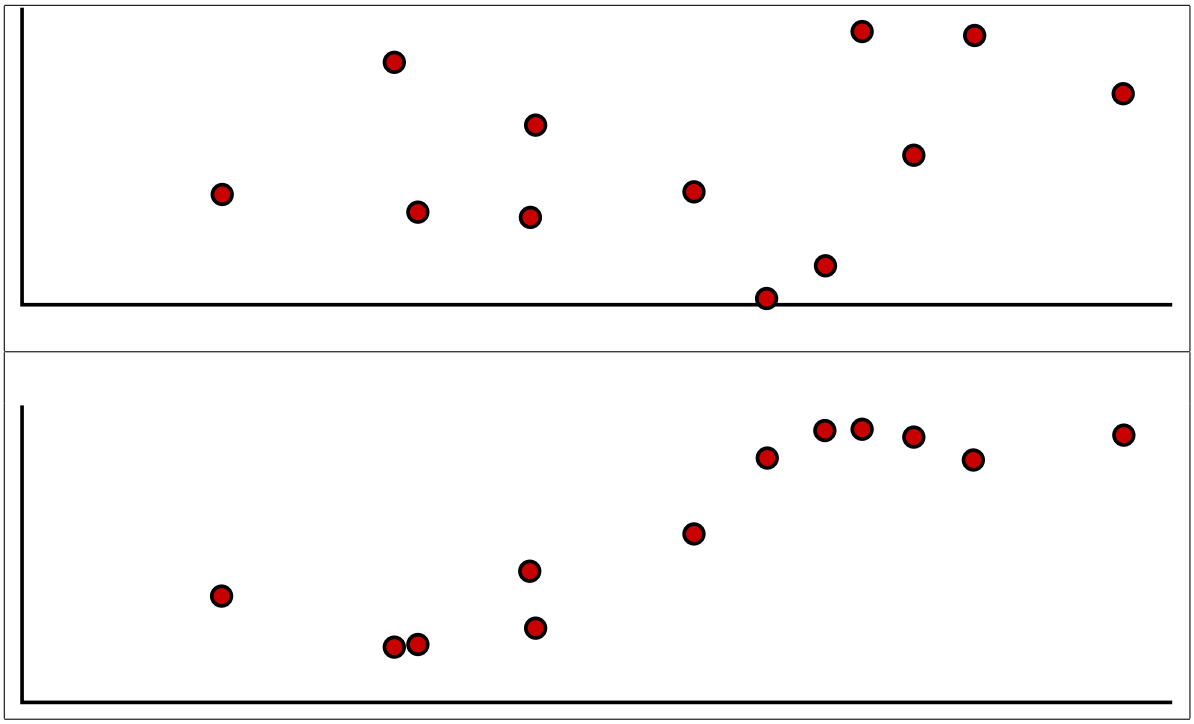


Figure 4.8: Plots of the values of a parameter for each of the quadrupedal animals in the motation database. The y-axis of each of the two plots is the value of the t_e parameter (see table 4.1) for each animal while the x-axis represents the logarithm of each animal's mass. The top plot shows the parameter values resulting from a standard inverse optimization while the bottom plot shows the results of a joint-inverse optimization. The parameters resulting from joint-inverse optimization are substantially better suited to regression. Note that distances along the x-axis are only an approximation to the similarity to the animals as measued by equation 4.17, so a perfectly smooth curve should not be expected in the lower plot.

4.5.1 Regression and Regularization Functions

The preceding discussion of joint inverse optimization was agnostic as to what particular regression function $R(A_i, \theta)$ was used to estimate the best vector of inverse parameters for an input animal. In any actual implementation, however, both the regression function $R(A_i, \theta)$ and the regularization function $r(\theta)$ will need to be defined. In my own implementation I have chosen relatively simple definitions for these functions based on radial basis functions with regularization based on quadratic smoothing [7].

Since I employ a regression function based on radial basis function, some method is

required to measure the similarity of the shapes of two different animals A_a and A_b. This measure of similarity is implemented as a distance metric $d(A_a, A_b)$ which yields 0 if $A_a = A_b$ and increasingly larger values the more different A_a and A_b are. While there is, of course, no 'true' measure of the similarity between the shapes of different animals, it is still important that this distance metric capture a notion of similarity in the form of animals which is reflected in the similarity of their motions. Assuming that A_a and A_b have the same topology, I have found the following distance metric to work relatively well:

$$d(A_a, A_b)^2 = \sum_i \left(\frac{l_{a_i}}{\sum_j l_{a_j}} - \frac{l_{b_i}}{\sum_j l_{b_j}} \right)^2 + 0.002(\ln(m_a) - \ln(m_b))^2 \tag{4.17}$$

where l_{a_j} and l_{a_j} are the lengths of the jth limbs in A_a and A_b respectively, and m_a and m_b are the total respective masses of A_a and A_b. Intuitively, this distance metric is formed by a weighted combination of the difference of the log-masses of the two animals and the difference in the lengths of their various limbs normalized by the total size of each animal.

Using this distance metric, the regression function is then defined in a manner similar to [93] as:

$$R(A, \theta) = \frac{\sum_i \theta_i e^{-\frac{d(A,A_i)^2}{d_{min}^2}}}{\sum_i e^{-\frac{d(A,A_i)^2}{d_{min}^2}}} \tag{4.18}$$

where $d_{min} = \min_i d(A, A_i)$. Here θ_i are the regression parameters directly associated with A_i (which in general need not be equal to ϕ_i).

Because this regression function is nonparametric, it may be prone to overfitting. To avoid overfitting a definition of a regularization function is also required:

$$r(\theta) = \sum_i \|R(A_i, \theta') - \theta_i\|_2^2 \tag{4.19}$$

where θ' are the regression parameters omitting the θ_i parameters associated with A_i, so this regularization function essentially computes a sum of leave-one-out errors,

Finally, the period p_i of an animal's gait cycle and the speed v_i at which it should move

are treated separately and determined by:

$$p_i = \alpha_p {m_i}^{\beta_p} \tag{4.20}$$

$$v = \alpha_v {l_{leg_i}}^{\beta_v} \tag{4.21}$$

In these equations m_i is the total mass of and l_{leg_i} is the average length of the legs of A_i.

Additionally, α_p and β_p are parameters used to model the relationship between an animal's mass and the period of its gait cycle, while α_v, and β_v are parameters modeling relationship between the length of the animal's legs and its speed.

4.6 Results

The previous chapter (chapter 3) gave a technique for synthesizing a cyclic gait for an animal without requiring any motion data for the animal. The inverse-optimization based approaches given in this chapter are effective at improving the realism of these synthesized gaits. In addition to allowing more realistic gaits to be synthesized, approaches such as those I have have given here and which are based on estimating the optimal inverse parameters for an animal have a further advantage of flexibility. In contrast to approaches based on kinematic retargeting, a vector inverse parameters makes no reference to the particular type of motion being synthesized and thus may be used to synthesize different types of motions or motions for different animals than those which the parameters were learned on.

In order to quantitatively compare my approach with other alternatives, table 4.2 shows how closely the result of a leave-one-out test matches the 'true' gait for several potential synthesis techniques.

The different methods tested in table 4.2 are as follows:

default The gait is synthesized using the method described in chapter 3.

kinematic interpolation A comparison method in which gaits in the motion database are directly interpolated weighted according to equation 4.18. The resulting motion will generally not satisfy the laws of physics, so a final spacetime constraints optimization is

method	**mean error**	**median error**
default	0.495	0.435
kinematic interpolation	0.451	0.387
naïve inverse interpolation	0.473	0.360
joint-inverse interpolation	0.327	0.245
(base inverse fit)	0.142	0.127

Table 4.2: The mean and median errors over the combined biped and quadruped motion databases of leave-one-out tests in which one animal was excluded and its gait synthesized based on the other animals. The errors are measured using metric given by equation 4.7. Note that the cheetah and emu were excluded from these statistics because the 'naïve inverse interpolation' spacetime constraints optimization failed to converge for them.

performed which attempts to match the kinematic interpolation as closely as possible while satisfying the laws of physics.

naïve inverse interpolation The gait is synthesized using inverse parameters interpolated using equation 4.18, but *without* any joint inverse optimization (i.e. each the inverse parameters for each animal are optimized independently).

joint-inverse interpolation The gait is synthesized using inverse parameters interpolated using equation 4.18 *with* joint inverse optimization.

(base inverse fit) For comparison, each gait is synthesized using the optimal independently-fit inverse parameters. In contrast to the other approaches listed here this one *does* make use of the target motion of the animal.

All of these comparisons are made with respect to the motions fit from video for each animal as described in section 4.3, and given over the set of all the bipeds (figure 4.2) and quadrupeds (figure 4.4) in the motion database, excluding the cheetah and emu as the optimization used by the 'base inverse interpolation' method failed to converge for these two animals.

As shown in table 4.2, the most accurate gaits are achieved by a synthesis based on regression of inverse parameters from a motion database after the parameters in the database

have been refined by joint inverse optimization (section 4.5). Somewhat surprisingly, this approach is superior even to a method based on a kinematic interpolation of the motions of similar animals, despite the fact that the kinematic method is substantially less flexible, and cannot easily be used, for instance, to synthesize gaits for the animal moving at a different speed or with a different gait than was interpolated from the database. Illustrations of the motions generated by leave-one-out syntesis using my approach are shown for a flamingo, elephant, and giraffe in figures 4.12, 4.13, 4.14. A further illustration of the motions associated with the various stages in the fitting process for a gazelle is given if figure 4.11.

Synthesis by regression on inverse parameters as found through joint-inverse optimization is also successful at synthesizing gaits for animals which are relatively dissimilar to anything in the motion database. As an illustration, figures 4.9 and 4.10 show the gaits synthesized for a number of extinct animals. Despite the shapes of these animals being relatively dissimilar to anything currently living, the synthesis procedure adapts the motion to the animal's shape and size, as illustrated on three members of the family dromaeosauridae in figures 4.15, 4.16, and 4.17 and for two members of the suborder ceratopsia in figures 4.18 and 4.19.

There are, unfortunately, a couple of situations where the method presented in this chapter does not perform as well. The first of these situations is when little data exists for a type of animal in the motion database. For instance, my motion database contains only three cats, and the leave-one-out motions synthesized for them show an error greater than for the other animals in the database. The second situation in which my method can perform poorly is when large extrapolations are used. For instance the motions synthesized for a paraceratherium (four times the mass of an elephant) or for *argentinosaurus huinculensis* (4.5 times the mass of an elephant) exhibit some bending of the legs which is probably not realistic. Extrapolating further to the motion of a *amphicoelias fragillimus* (20 times the mass of an elephant) yields clearly unrealistic results. Nevertheless, when used on animals with a size near those in the motion database, the method given here performs well, even if the shape of the animal itself is relatively unlike anything currently living.

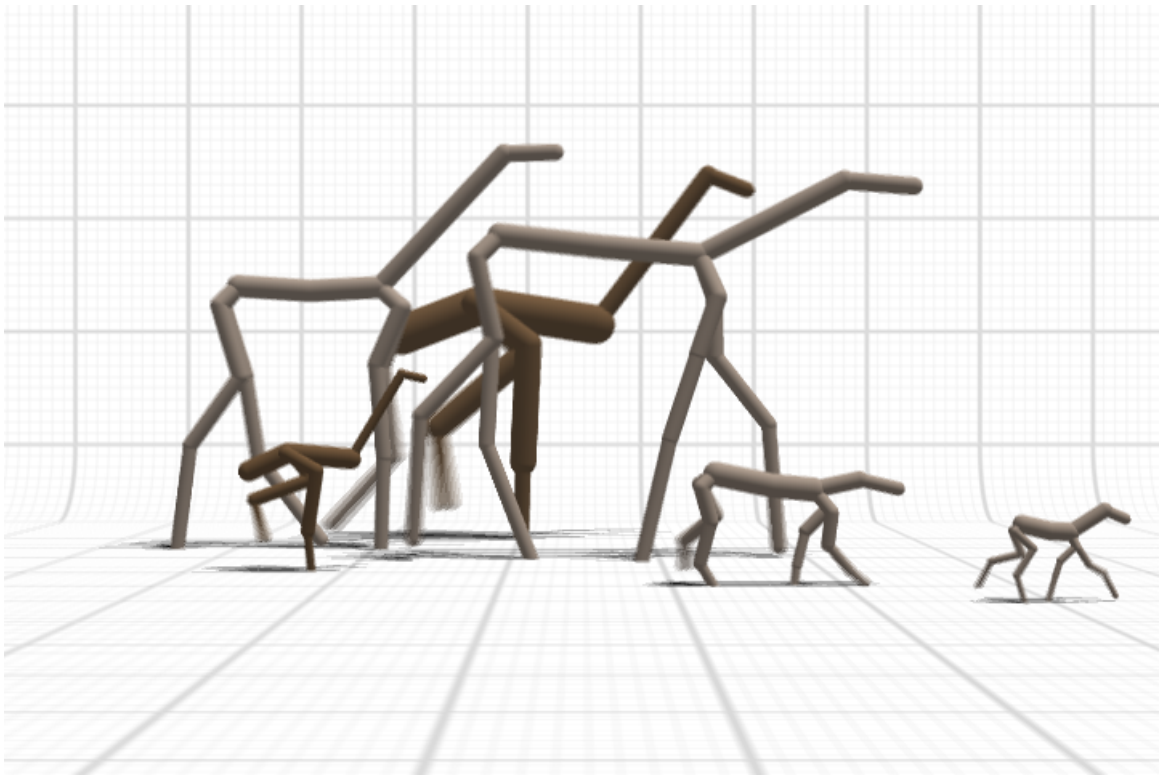


Figure 4.9: Synthesized gaits for a number of extinct creatures. From back to front appear a north island giant moa, aepycamelus, hemiauchenia, bush moa, phenacodus, and blastomeryx.

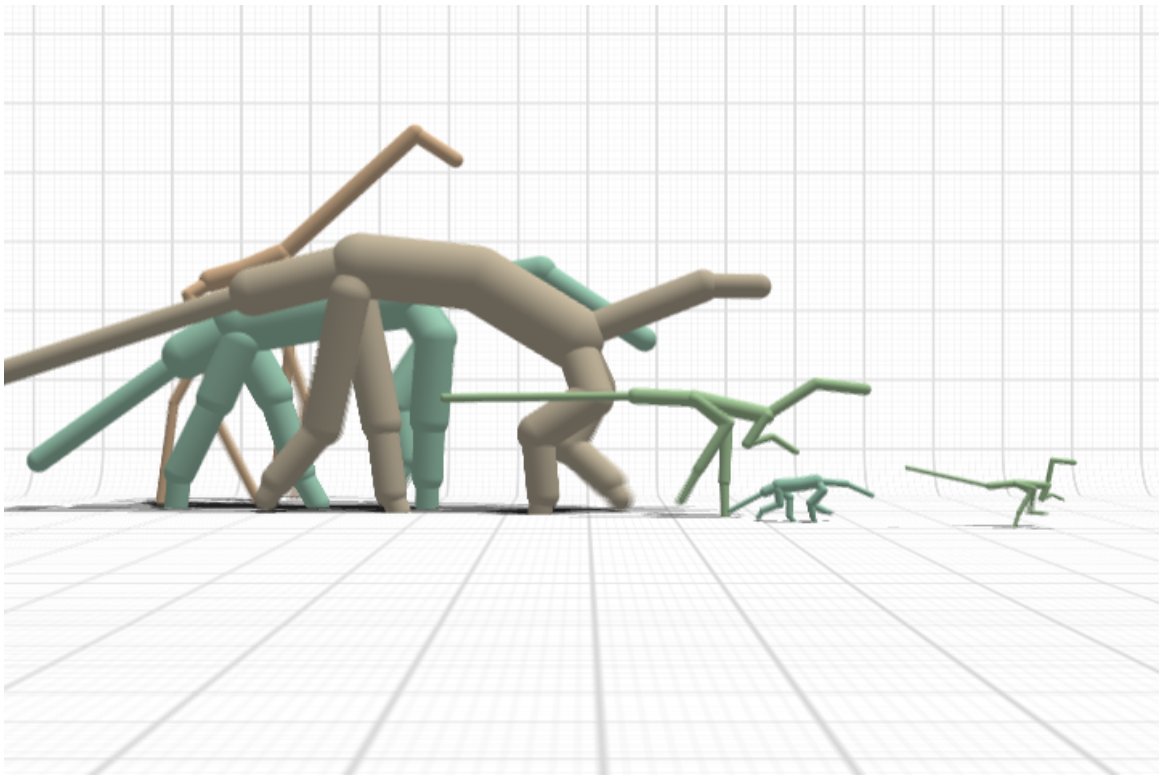


Figure 4.10: Synthesized gaits for a number of dinosaurs. A giraffe is shown in the back to provide a sense of scale. From back to front the dinosaurs appearing here are triceratops, stegosaurus, utahraptor, protoceratops, and velociraptor.

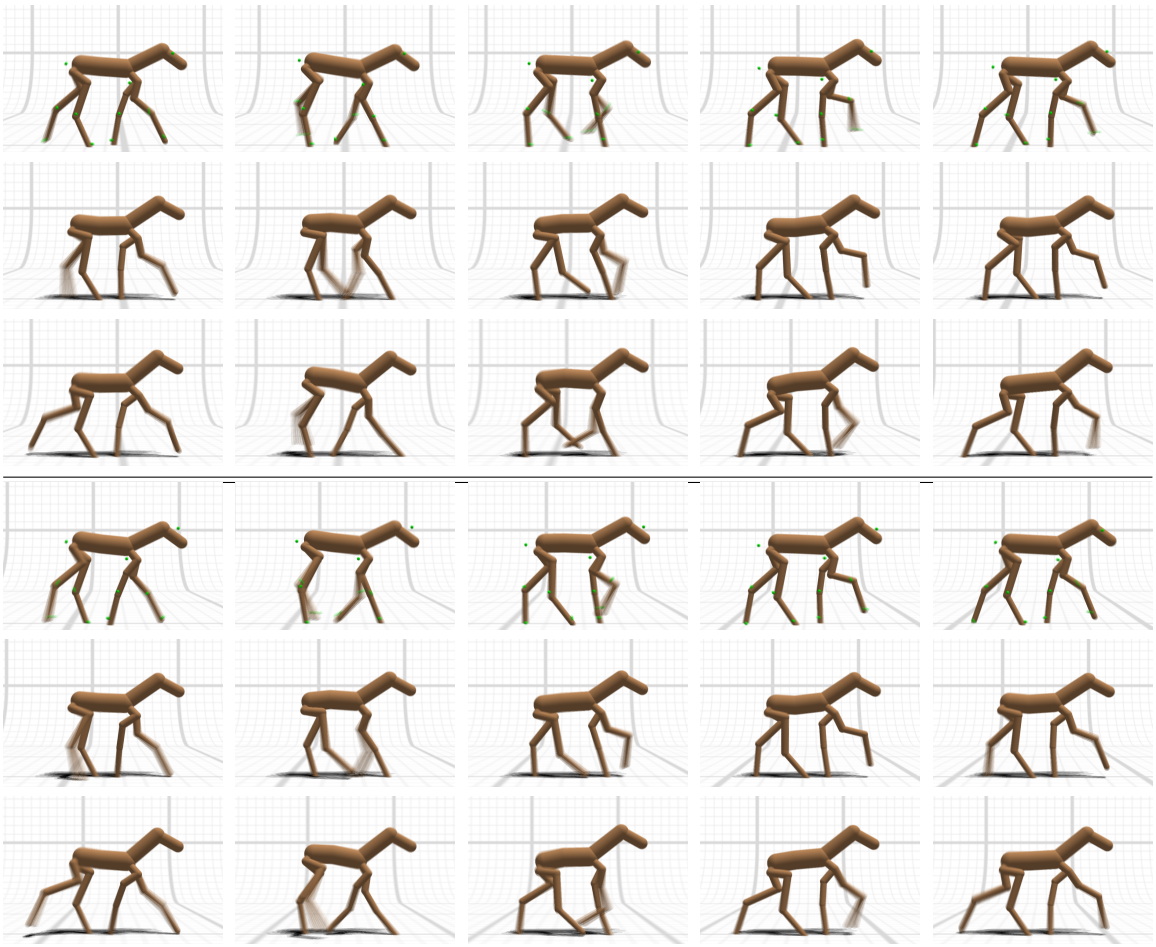


Figure 4.11: Animation frames for three different fitting stages of from an a gait for a Thomson's gazelle. From top to bottom: fit directly to points extracted from video, direct inverse fit, and leave-one-out resynthesis after joint inverse optimization.

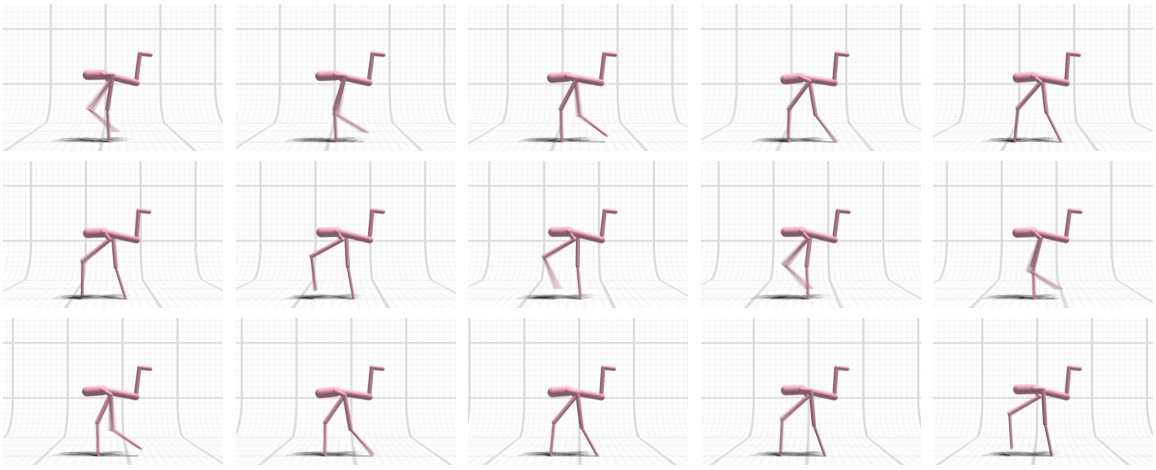


Figure 4.12: A leave-one-out gait synthesized for a greater flamingo.

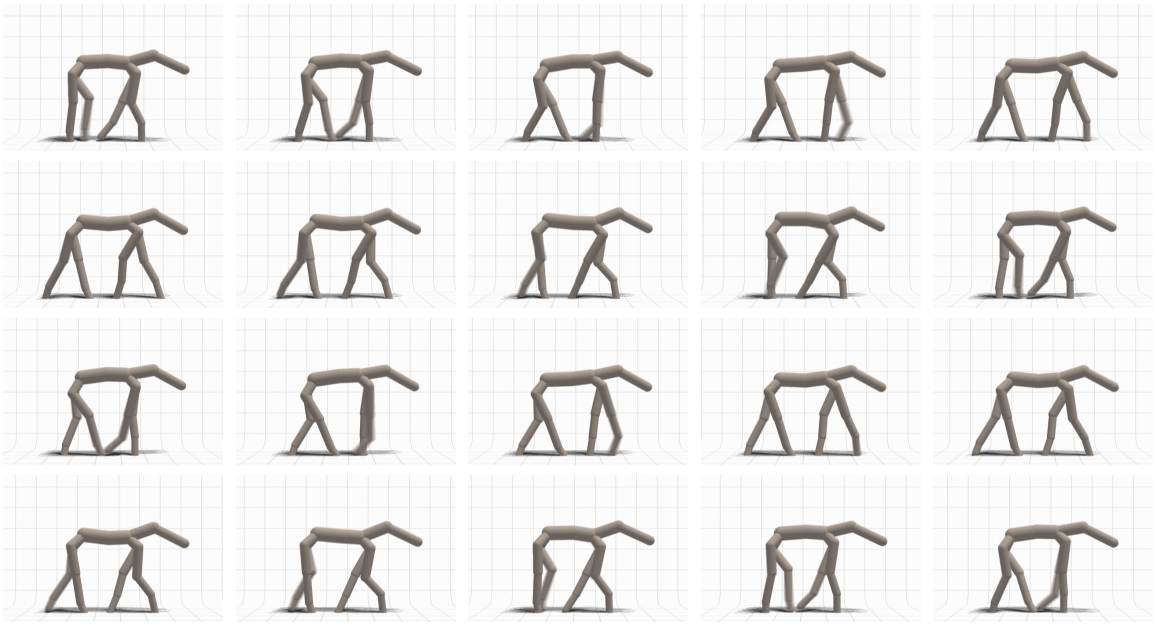


Figure 4.13: A leave-one-out gait synthesized for an elephant.

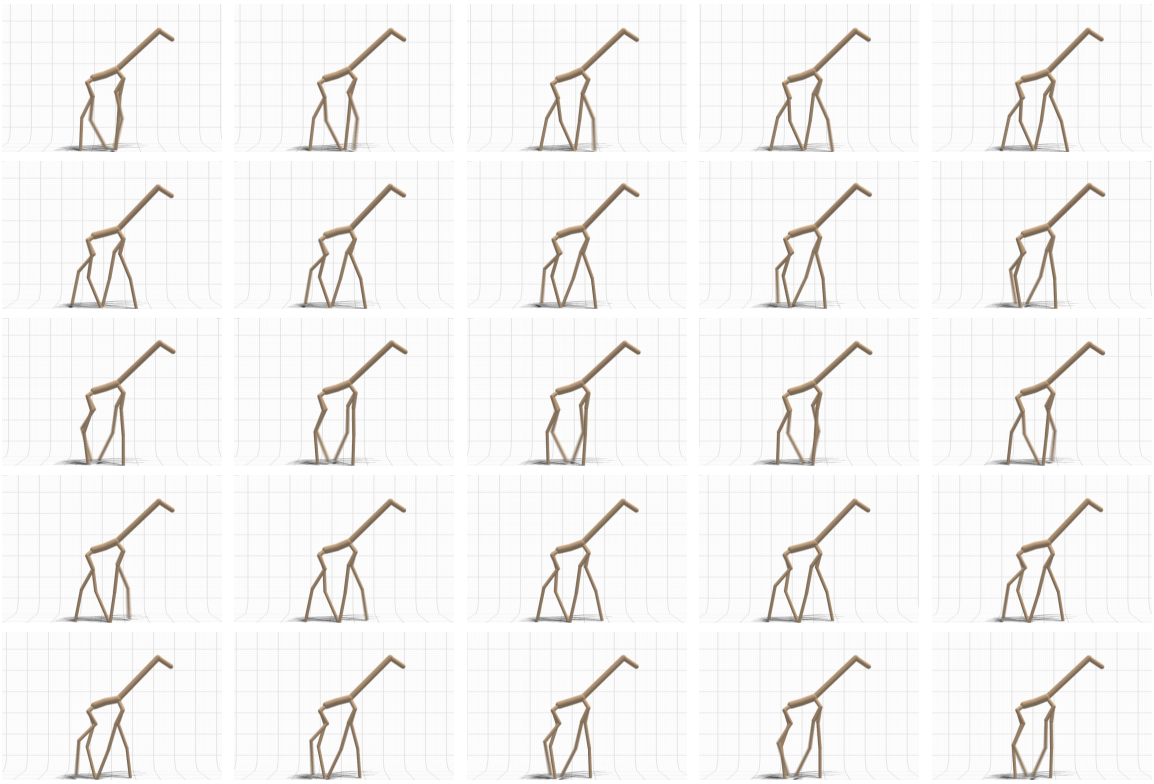


Figure 4.14: A leave-one-out gait synthesized for a giraffe.

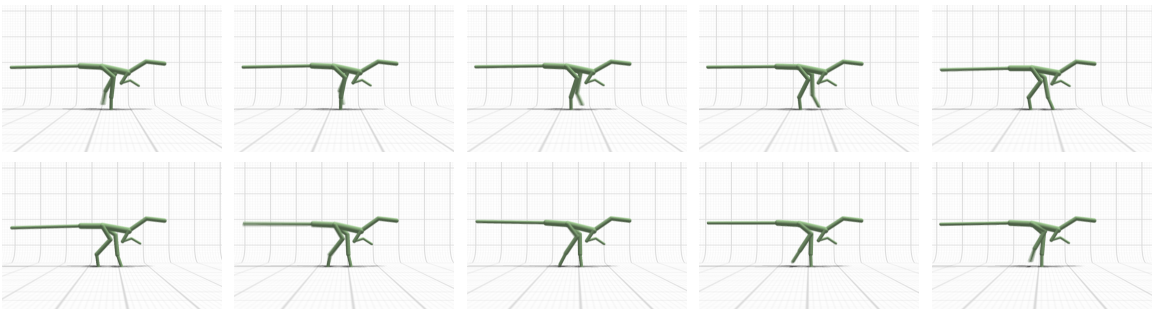


Figure 4.15: A gait synthesized for a utahraptor. Compare with the gaits of the smaller deinonychus (figure 4.16) and velociraptor (figure 4.17).

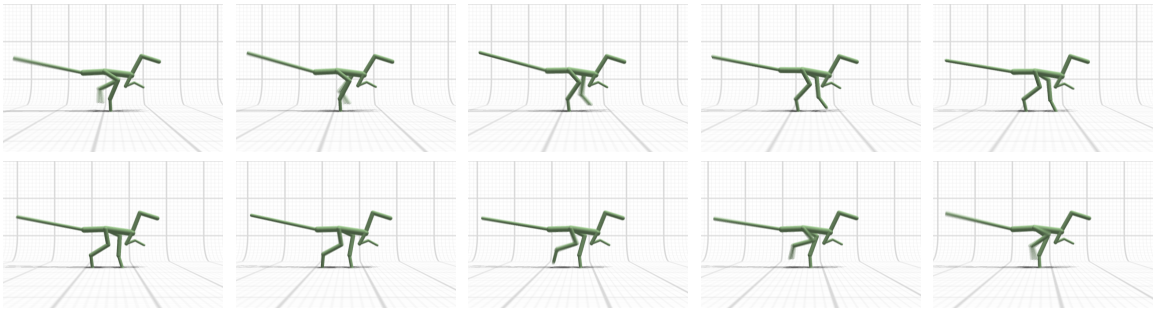


Figure 4.16: A gait synthesized for a deinonychus. Compare with the gaits of the larger utahraptor (figure 4.15) and smaller velociraptor (figure 4.17).

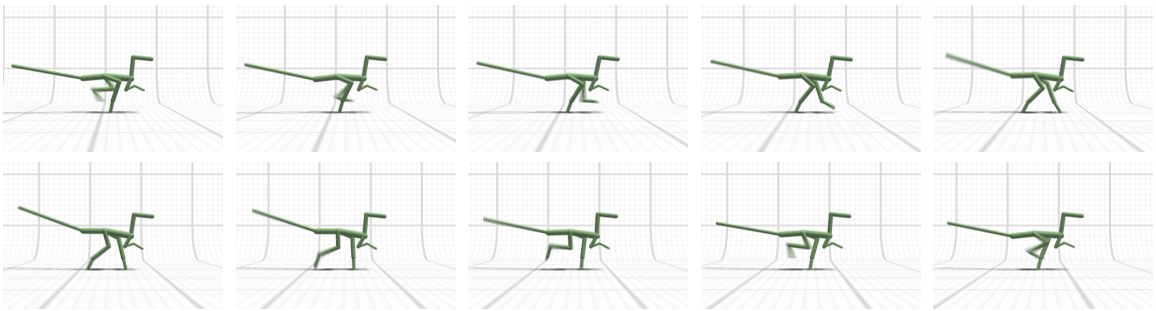


Figure 4.17: A gait synthesized for a velociraptor. Compare with the gaits of the larger utahraptor (figure 4.15) and deinonychus (figure 4.16).

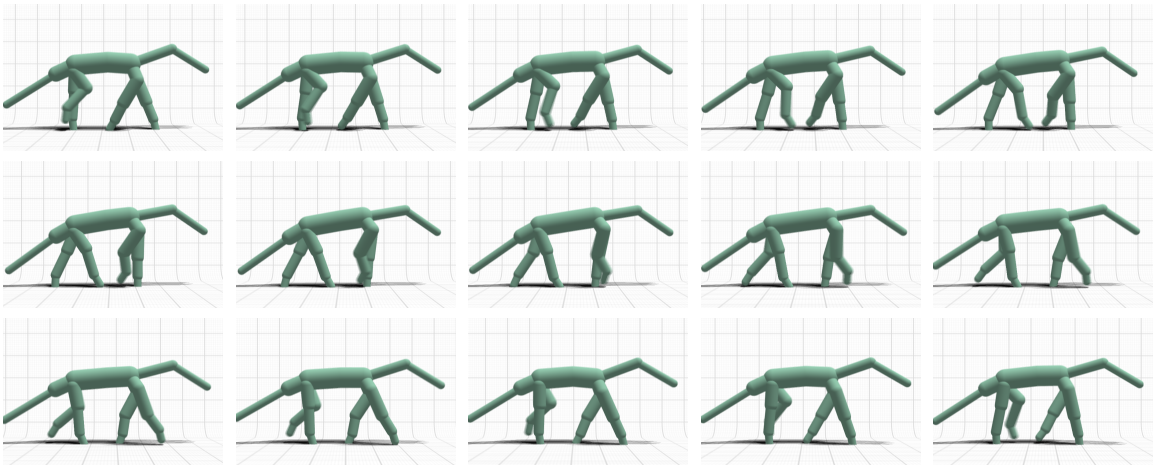


Figure 4.18: A gait synthesized for a triceratops. Compare with the gait of the smaller protoceratops (figure 4.19).

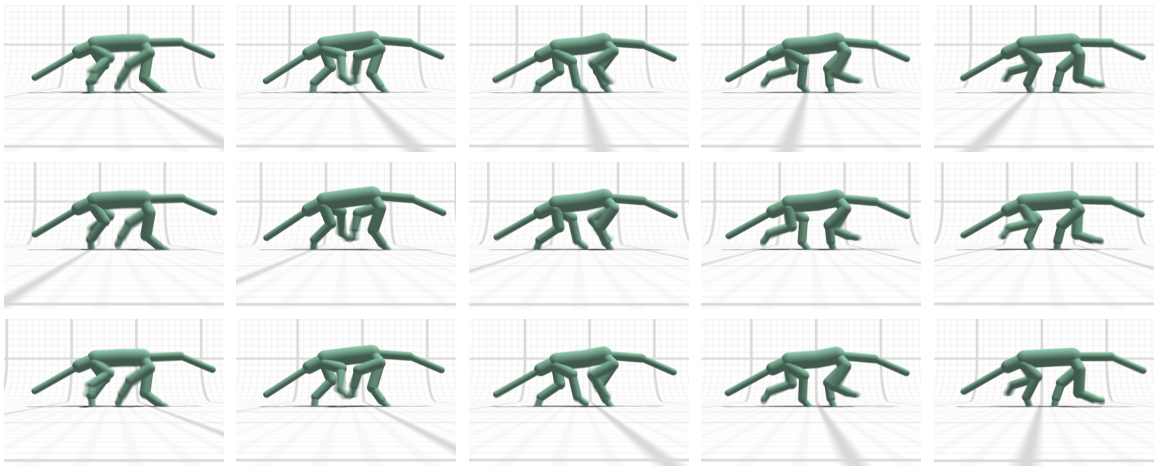


Figure 4.19: A gait synthesized for a protoceratops. Compare with the gait of the larger triceratops (figure 4.18).

Chapter 5

MOTION CONTROLLER SYNTHESIS

Although gait generation is perhaps the most obvious core application for animal locomotion synthesis, cyclic gaits account for a relatively small fraction of the variety of motions exhibited in nature, even when the context is restricted to just locomotion. In reality an animal can exhibit a range of motions including turns, gait transitions, and others depending on its environment and high-level goals. Furthermore, restricting motions to cyclic gaits precludes any real-time applications (such as video games) where the animal's motions must adapt in real-time to conform to a user's commands.

In this section I will present a technique for automatically creating controllers for an animal which can be used to generate interactive locomotion animations in real time. The input to this method consists of a small amount of information about an animal similar to that required for gait synthesis. In particular the approach requires the shape (skeleton) of the animal, the gaits which it can perform, and a model of the task or tasks for which the controller will be used. From this input data, the approach attempts to efficiently synthesize a physically realistic controller which can adeptly perform the required tasks.

The most fundamental choice to be made in constructing a controller is what the representation of this controller will be. There are two aspects which must be included in this representation: Which motions are possible and what choices as to its future motion are available to a character at any time. The approach I take is to employ a structure known as a *motion graph* [37].

Motion graphs are a common tool for designing interactive character controllers by organizing motion data into a graph-like structure. The branches in this graph correspond to points where a choice exists as to what the next moment in the animation will be, and by carefully picking the 'correct' branch to take at each point the animal can perform different tasks. For instance, a controller designed to follow a given direction would consist of various

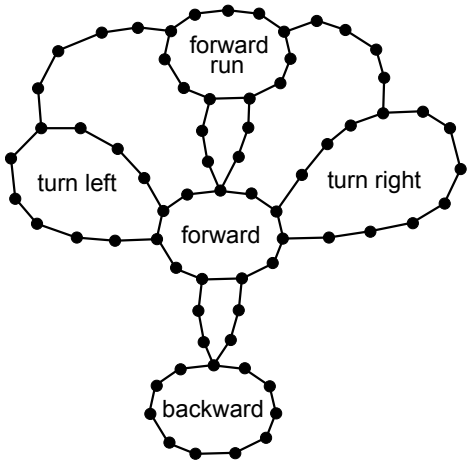


Figure 5.1: A simple example of a hypothetical motion graph. This graph allows three different cyclic motions (walk, run, and backward walk), transitions between these motions, and right and left turns.

connected walks and turns, and the way in which these walks and turns interconnect describe when, for example, the animal could switch from a straight walk into a turn (figure 5.1).

Since I do not assume access to any pre-specified information on how the animal moves, there are several challenges which must be overcome to accomplish this. Firstly, not all combinations of motions into a graph will create controllers which perform a task equally well. Indeed, it appears that even that task of selecting a set of motions from an existing database to create a controller is difficult to achieve [45]. On the other hand, the algorithm must also contend with the difficulty of actually creating the individual motions from which the graph is built. This is particularly difficult because new motions are most easily created by modifying similar existing motions, but controllers are are often best constructed from a highly dissimilar set of motions. Thus the choices needed to quickly build highly effective controllers must be balanced against possible difficulties in actually creating these motions.

I approach this problem by first creating a measure of the quality of a controller which unifies the objective functions used in spacetime constraints and reinforcement learning-based approaches and which encompasses both the quality of the individual motions and how adept the controller is at performing its user-specified tasks. This is combined with a function that quickly generates rough guesses at new motions which might be added to an

existing controller, and also estimates how likely it is that a final physically valid motion can actually be generated from each guess. Furthermore, each new added motion is used to improve the quality of future guesses, thus eventually allowing new motions which might not have been possible initially. These components allow the graph of motions that define a controller to be incrementally built in an automatic and efficient manner.

I'll begin the remainder of this chapter by providing a brief overview of the related literature. Following this, I'll show how the cost functions used in previous chapters to measure the optimality of single motions can be extended to measure the optimality of entire motion controllers. Using this definition of optimality I will then describe the process by which a motion graph is constructed, followed by some results generated by this approach.

5.1 Background

Locomotion behaviors are challenging to create because they must both nimbly respond to varying tasks and faithfully synthesize motions that accomplish them. Probably the most common way to proceed in order to achieve this is to rely on motion data [8, 85], motion graphs [41, 2, 37], or other parametric structures derived from motion data [66, 26, 36]. A further recent research trend is to use reinforcement learning to automate the task of selecting which data clips to play at runtime in order to best accomplish user-specified tasks [10, 42, 40, 75, 52, 51]. This approach is typically used in conjunction with controller representations which rely on motion data, and thus these methods explain how to compose responses by concatenating motion sequences, but they do not indicate how the motions themselves should be created, instead taking these motions as a pre-specified input.

Because of their predictability and generally speedy runtime performance, the controllers routinely employed in computer games are of this kinematic variety and fundamentally rely on pre-recorded motion data. Although most of these approaches are limited to directly playing back pre-recorded data, one notable exception to this is given in [27]. In this approach an artist designs motions using an interface that explicitly aids in making these motions retargetable to a wide range of different creatures. Although both this method and mine can construct controllers for many different creature types, they differ in two primary ways. Firstly, although the controllers generated with my approach are kinematic,

any motions generated by these controllers are guaranteed to obey the laws of physics. Secondly, the technique I present does not rely on a database of artist-generated motions but instead creates the motions directly from the shape of the animal.

An alternative approach, and one which appears to be more promising when pre-captured motion data is not available, is to construct dynamic controllers which attempt to synthesize motion for a character within a physics simulator. Approaches of this sort [16, 69, 11, 89, 57] leverage rapid advances in the design of control strategies for walking, running, and other motion primitives [63, 29, 28, 87, 90, 13, 12, 15] to create these controllers. Recent advances have also enabled dynamic quadruped controllers [14] with a wide range of simulated gaits and skills. Although the variety and quality of the motions generate by this technique are impressive, it is not a general-purpose controller synthesizer that can be applied to an arbitrary animal since, aside from being limited to quadrupeds (demonstrated on a dog), constructing the controllers requires either motion capture data or a careful tuning of parameters. The approach presented here differs from existing ones in that it requires no data or manual tuning, yet can be applied to a wide range of different animals.

My approach to motion controller synthesis is based on trajectory synthesis with space-time constraints. In particular, the methods for synthesis of animal gaits presented earlier and techniques to synthesize short motion transitions [9] hint at the possibilities for automatic computation of locomotion behaviors. In the remainder of this section, I will show how to go beyond the computation of steady-state animal gaits by synthesizing motions with turning and varying gait transitions. These motion optimizations can then be combined with an outer loop that builds and chooses which motions to generate in order to build a locomotion behavior.

5.2 Locomotion Controller Construction

Before describing my approach for automatically generating animal locomotion controllers, it's useful to provide some basic definitions for terms which will appear later. As before, the input to this method is an animal specified by a kinematic skeleton, so that a static pose for the animal can be represented by a vector of DOFs (figure 2.1). An animation for an animal is specified by a sequence of DOF vectors, one for each frame (figure 2.2).

Now, however, it becomes useful to distinguish between pre-computed animations of a fixed length and animations generated on the fly by the locomotion controller. The former (fixed length) motions will thus be referred to as *motion clips*, denoted M. An individual frame in a motion clip, f_i, will be denoted by $M(f_i)$. It will be occasionally be useful to index into a motion by a continuous time parameter instead of a discrete frame parameter, and this bill be notated with $M(t)$ and computed by linearly interpolation from the pair of frames surrounding t.

A motion graph consisting of many interconnected motion clips will be denoted with with $\mathbf{G}$. The nodes in this graph are equivalent to frames in a motion, and $\mathbf{G}(f_i)$ will be used to refer to such a node. The edges in a graph represent frames which can be transitioned between, and also store the relative 2D transformation which the root of the character undergoes during the transition. Finally, throughout the controller construction process it will be it useful to create approximations of various quantities. Such approximations will be signified by a $\widetilde{\text{tilde}}$.

5.2.1 Controller Optimality

The goal of my approach is to create a real-time locomotion controller which is both biomechanically efficient and which can adeptly satisfy the user's commands. This controller is constructed from a set of motion clips, $M_1, \ldots, M_n$ interconnected into a single graph $\mathbf{G}$. Each branch in this graph corresponds to a point where the character has a choice in its future motion, the goal being to choose the particular sequence of motions which best complies with the user's commands.

To create effective controllers a method is required to judge how 'good' a candidate controller is. This is analogous to the way in which an objective function determining how 'good' a gait is forms a critical (perhaps *the* critical) component of the gait synthesis techniques presented in chapters 3 and 4. Now, however, this objective function must be applicable to the the synthesis of individual motions clips, the construction of a motion graph out of these clips, and the use of this graph to generate motion at runtime in response to a user's commands. These new requirements placed upon the use of the objective function

necessitate moving beyond the notion of a cost function as used in standard spacetime constraints optimizations (equation 3.1).

Before coming to the full definition of the cost function it is useful to begin by considering how to generate single motions offline. The technique used for this is based on the spacetime constraints formulation detailed in the preceding chapters 3 and 4. This approach specifies a motion as the minimizer of a large constrained optimization problem and represents the objective function as a sum of per-frame costs and specifies the constraint functions independently for each frame. Assuming that the desired motion consists of n different frames, recall from equation 3.1 that this optimization has the form:

$$\begin{aligned} M^* = \underset{M}{\operatorname{argmin}} \; & \frac{1}{n} \sum_i cost(M(f_i)) \\ \text{s.t. } & g_j(M(f_i)) = 0 && \forall j, i \\ & h_k(M(f_i)) \leq 0 && \forall k, i \end{aligned}$$

where $cost(M(f_i))$ is taken to be representative of the effort exerted by the character at frame f_i and the constraint functions g_j and h_k enforce that the resulting motion satisfy the laws of physics (refer back to chapter 3 for more detailed explanations of these terms).

Although this formulation can work well for synthesizing single motions, it does not provide any means by which user commands can be incorporated. Thus the first step in defining a cost function suitable for motion controller synthesis is to generalize the spacetime constraints cost function $cost(M(f_i))$ to one which does allow a mechanism to alter the resulting motion in response to user commands. For the moment, however, it's simpler to restrict attention only to the offline synthesis of individual motions clips, in which case the 'user commands' are not to be thought of as provided in real-time, but merely as prespecified parameters which allow control over the type of motion to be generated.

In order to incorporate user commands into the motion synthesis process it is necessary to provide a way in which such commands can be represented. To this end, a vector of *task parameters* $\theta(f_i)$ is defined for each frame $M(f_i)$. The definition of the task parameters is task specific, and provides the means by which the user can control the character. For

example, in a controller which allows the character to be steered to walk in a desired direction, the task parameters would simply include the character's desired heading. The cost function used by spacetime constraints is then altered to penalize deviations from performing the desired task at each frame.

Since this change makes the cost function dependent not only on the state of the character $M(f_i)$ at each frame but also on the user's task parameters $\theta(f_i)$ at that frame, the cost function must be written as a function of these two vectors at each frame as $cost(M(f_i), \theta(f_i))$. A simple but flexible way of defining this new cost function represents the total cost at each frame as the sum of two components:

$$cost\big(M(f_i), \theta(f_i)\big) = cost_b\big(M(f_i)\big) + cost_t\big(M(f_i), \theta(f_i)\big) \tag{5.1}$$

Here $cost_b\big(M(f_i)\big)$ is the *biomechanical cost* inherent in the character's motion at frame fi and is equivalent to the gait cost function in 3.1. $cost_t\big(M(f_i), \theta(f_i)\big)$ on the other hand is the *task cost* penalizing frames in which the character's state $M(f_i)$ is not in accordance with the task parameters $\theta(f_i)$. Details of some specific task parameters this method has been tested with can be found in section 5.3.

This formulation allows the synthesized motion to be altered by defining different values for the task parameters $\theta(f_i)$ at each frame. While this allows the incorporation of the user's commands, the motion itself must be re-synthesized offline each time the task parameters are changed. Nevertheless, the ability to synthesize a range of different motions allows, in principle, a diverse vocabulary of different motions to be synthesized and interconnected into a motion graph. The next step in defining the cost function for the locomotion synthesis process is then to extend the preceding cost function for motion clips to be applicable to entire motion graphs.

The crucial difference between a motion graph and a motion clip is that a motion graph in in general does not correspond to a single fixed motion, but can rather be used to synthesize a variety of different motions in response to different sequences of task parameters. Doing so involves moving from a view of a motion as something taking place over a number of frames occurring at with fixed as pre-specified times to a more dynamical view. Mathematically

speaking, this most naturally leads to a formulation of a controller as a *Markov decision process* (MDP).

Markov decision processes are a relatively general mathematical framework for dealing with optimally making choices in time-varying systems. In an MDP each possible state of a controller is represented as a point in a *state space*. In this project, a point in this state space, s, is defined by a tuple of the form (f, r, θ) where f is a frame in the motion graph $\mathbf{G}$, r is a matrix giving the global 2D position and orientation of the character, and θ is some setting of the problem's task parameters.

Since a user controls a character by changing the task parameters as the character moves, the formulation can no longer be restricted to a fixed sequence of task parameters. Instead it is necessary to specify a dynamical model by which these task parameters are expected to change over time. Intuitively, this model can be thought of as specifying the expected behavior of the user as they control the character. Although in principle this task model could take any form, in my implementation I only use Markovian models which specify a probability $\mathrm{P}\left[\theta(f_{t+1})|M(f_t), \theta(f_t)\right]$ of the user selecting task parameters $\theta(f_{t+1})$ for the next frame if the task parameters in the current frame are $\theta(f_t)$ and the character's current state is $M(f_t)$. Although this is a restricted model of a user's expected behavior, I have found it to be sufficient for controller synthesis, and it can also be easily learned from examples of real user behavior [52].

In addition to the aforementioned state space, an MDP requires the specification of a *transition function*, of which this model of how the task parameters are expected to change over time forms one part. In short, the transition function defines the possible ways in which one state, $s = (f, r, \theta)$, can change to an new state $s' = (f', r', \theta')$ in the next frame. It is therefore by means of the transition function that the time-varying behavior of the controller is specified.

In the particular MDP used for the synthesis of locomotion controllers in this project, this transition function updates the state s to s' by changing f to f', altering r by the transformation associated with the edge in $\mathbf{G}$ between f and f', and stochastically updating θ according to $\mathrm{P}\left[\theta'|M(f), \theta\right]$. Since this transition depends on both f and f', and because the motion graph $\mathbf{G}$ often defines more than one motion state (f') which can be transitioned

to from f, the character can be viewed of as having a 'choice' as to which motion state to transition to next. This choice, however, only extends as far as the selection of which edge to take in the graph at each time, and once an edge is chosen the change of the root orientation r and task parameters θ is viewed as 'outside the character's control' (although the character's choice of f' can take into account the changes in r and θ resulting from this choice).

The final component of an MDP is a *cost function* giving the cost incurred by transitioning from state s to s' as $cost(s, s') = cost(M(f), \theta)$. The definition of this cost function itself is identical to that indicated previously in equation 5.1, but its use is somewhat different since as it no longer applies to a single motion of pre-defined length. The goal of an MDP controller such as the one to be used for locomotion synthesis is to pick which edges in $\mathbf{G}$ to follow so as to minimize the cost of the corresponding state transitions, given the current values of the task parameters (whatever the user may have then currently set to).

Because there is no way to know the sequence of task parameters a user will choose in advance, the objective function in equation 3.1 cannot be directly used to evaluate the cost of using a controller built from a particular graph. More specifically, it is no longer feasible to compute the sum of the cost at each of a sequence of motion states and task parameters because no such fixed sequence exists in the controller setting. Still, since the version of the objective function given in 3.1 gave the average per-frame cost of a motion, the objective function used in controller synthesis should reflect an analogous concept.

To achieve a definition for the objective function which is analogous to that used in the spacetime constraints optimization, the objective function is altered so that instead of being defined by the average per-frame cost of a motion, it represents the expected value of the average per-frame cost incurred while using the controller. This expected cost depends on the *policy* π which dictates which edge the $\mathbf{G}$ should be followed from each possible combination of f and θ. Evaluating a policy at a point in state space yields a new point in state space $s' = \pi(s)$. I make the standard choice that this policy be required to be an *optimal policy* π^* which minimizes the associated expected cost over the space of all possible policies.

In order to numerically calculate the expected average performance of a controller the

standard approach is to use reinforcement learning [72]. In my case this is done with a two-step process: First an optimal policy π^* is calculated which determines which edge should be taken in the graph for each possible combination of node and task parameters. This can be efficiently computed by calculating a *value function* over the combined space of graph nodes and task parameters. Actually calculating this value function is a well-studied task, and details can be found in any of a number of character animation papers [42, 40, 75, 52, 51].

Once the optimal policy π^* has been determined, an *influence function* is computed which gives the probability with which a character moving according to π^* will be found in any particular state, taken in the limit as the controller is run for an infinite amount of time. Intuitively, this influence function captures the notion that some motions may be used more often than others, and thus costs for these motions should likewise matter more. This approach has proven successful in creating compact controllers from a database of pre-captured motions [45]. This influence function is needed to calculate the expected cost of the policy π^*, as this cost depends on which states π^* visits more frequently.

Since this influence function is defined over the entire state space of the controller, and because this state space contains some continuous dimensions (for instance the global translation and rotation of the character's root position) it can only be represented approximately. I take what is probably the simplest way to do this and approximate the continuous distribution by a set of discrete points in the MDP's state space. This results in a number of samples $s_1, \ldots, s_n$ distributed according to the influence function.

To compute these samples, $s_1, \ldots, s_n$ are initially distributed evenly over the state space, and then repeatedly advected according to the optimal policy π^*. This is done by, at each point and iteration, using π^* to determine an which edge in $\mathbf{G}$ to follow and then following the MDPs transition function for this edge to arrive at a new point in state space. After a sufficient number of iterations the state points accurately sample from the influence function. The expected value of the average per-frame cost of using the controller with the given task model is then calculated as a simple sum over these sample points:

$$\frac{1}{n} \sum_{s_i} cost(s_i, \pi^*(s_i)) \tag{5.2}$$

where $cost(s_i, \pi^*(s_i))$ is a notational convenience expressing the cost $cost\big(M(f_i), \theta(f_i)\big)$ where $M(f_i)$ is the frame in $\mathbf{G}$ associated with the state s_i.

The fact that more frequently visited states should have a greater influence on the cost of a controller is captured by the fact that the sample points will be more densely concentrated around these states. Furthermore, note that when solving a spacetime constraints problem to generate a motion that these samples only alter the objective function in equation 3.1, leaving the constraint functions unaltered. This means that this sampling procedure does not impact the physical validity of the generated motions. Nevertheless it in still necessary to consider these constraints in the context of a full motion graph.

So far this accounts for how the objective function in equation 3.1 generalizes from motion synthesis to controller synthesis. It is still necessary to address the constraint functions. Fortunately the answer here is simple. Each motion M of which $\mathbf{G}$ is comprised is a minimum solution to equation 3.1, and thus satisfies the constraint functions. It merely remains to ensure that the constraint functions are also satisfied at the junctions between motions. It is sufficient to simply require that when adding a new motion M to $\mathbf{G}$, the first and last pairs of frames in M each correspond to a pair of consecutive frames in $\mathbf{G}$. This ensures that the motions seamlessly stitch together and that all animations generated by a controller using $\mathbf{G}$ will be physically valid.

5.2.2 Controller Generation

Equation 5.2 allows the merits of one motion graph to be judged against another, but it remains to actually construct a graph which minimizes this cost. An effective method for doing so is to construct such a graph incrementally, where $\mathbf{G}_{i+1}$ is created by generating a new motion M_{i+1} and adding it to $\mathbf{G}_i$. There are two problems which must be addressed to accomplish this. First it must be possible to construct a wide range of potential motions which can be used to augment $\mathbf{G}_i$, after which one particular motion must actually be created and added to $\mathbf{G}_i$ so as to best improve the resulting controller. It is also worth noting that although these new motions are created and added one at a time, each motion is chosen and synthesized based on its context within the entire $\mathbf{G}_i$ (see appendix C).

Searching for which motion will best augment $\mathbf{G}_i$ is problematic because it is difficult to tell how a graph will be improved by adding a new motion without actually solving an expensive spacetime constraints problem to generate the motion. To circumvent this I instead kinematically generate a quick approximate motion $\widetilde{M}$ which serves a dual purpose both as an efficient-to-compute guess at the result, M, of a spacetime constraints optimization, and as the initialization to the same spacetime constraints problem should it actually be computed in full. These approximate motions are then used to efficiently compute an estimate to equation 5.2 for $\mathbf{G}_{i+1}$.

Pseudocode for the optimization process which iteratively constructs a controller graph $\mathbf{G}_i$ is given in figure 5.2. Roughly speaking, this involves four sub-components:

1. A method which allows a wide range of individual motions to be generated.

2. An efficient way to generate $\widetilde{M}$.

3. A way of quickly estimating how a candidate motion will improve $\mathbf{G}_i$.

4. A search over the space of possible new motions to determine the single next motion to solve in full and add to $\mathbf{G}_i$.

I will discuss each of these components in turn.

Motion Optimization

Each motion M_i is generated by starting with a 'guess' initialization motion $\widetilde{M}_i$, which is used as the initialization for a spacetime constraints optimization. The result of this optimization is both physically realistic and energy efficient. For the moment the initialization motion $\widetilde{M}_i$ will be taken for granted in order to focus on how spacetime constraints optimizations are used to generate the motions which constitute a controller.

The spacetime constraints formulation used for this is based on the methods presented earlier in chapter 3. Unfortunately, these approaches by themselves only generate cyclic 'straight run' gaits, and thus must be modified in order to produce the wide range of

Require: $\mathbf{G}_0$
while not done **do**
  $minCost \leftarrow \infty$
  $S^*, \sigma^* \leftarrow$ none, none
  for all S_j **do** ▷ find next optimization to perform
    search σ_k
      $\widetilde{M} \leftarrow$ initialization motion for S_j, σ_k
      $\widetilde{\mathbf{G}} \leftarrow \mathbf{G}_i \cup \widetilde{M}$
      $p \leftarrow \widetilde{P}(\widetilde{\mathbf{G}}, \sigma_k)$
      $c \leftarrow (1-p) \cdot cost(\mathbf{G}_i) + p \cdot \widetilde{cost}(\widetilde{\mathbf{G}}, \sigma_k)$
      if $c < minCost$ **then**
        $minCost \leftarrow c$
        $S^*, \sigma^* \leftarrow S_j, \sigma_k$
      end if
    end search
  end for State $\widetilde{M_{i+1}} \leftarrow$ initialization motion for S^*, σ^*
  $M_{i+1} \leftarrow S^*(\widetilde{M_{i+1}}, \sigma^*)$ ▷ perform optimization
  $\mathbf{G}_{i+1} \leftarrow \mathbf{G}_i \cup M_{i+1}$
end while

Figure 5.2: Pseudocode for the controller generation algorithm. The outer loop adds new motions to each consecutive $\mathbf{G}_i$ one at a time. The middle for-loop searches over the different types of motions that might be added (for instance adding a new cyclic gait versus adding a new turning motion). The inner search determines the optimal values defining particular qualities of the new motion in consideration for being added.

motions needed for a useful controller. At a high level, this is achieved by exposing some parameters of the optimization and then searching for the values of these parameters which are expected to generate the most useful new motion.

Each new motion M_{i+1} is used to augment the graph $\mathbf{G}_i$. Although the algorithm's structure supports any number of ways of augmenting a graph, I use two primary methods for achieving this: Adding a new motion providing a transition between two existing states, and creating a new cyclic motion and connecting it to $\mathbf{G}_i$. To represent each of these two possible modifications I define a pair of *optimization templates*, denoted S_1, S_2 (see figure 5.3). Each optimization template further exposes a space of parameters such that any setting of these parameters for S_j, denoted σ_j, yields a concrete optimization problem

which may be solved to generate a new motion $M' = S_j(\widetilde{M}, \sigma_j)$. Which parameters are exposed to be varied and which are fixed depends on the nature of the controller being generated, but the following are the ones used in creating the locomotion controllers shown here:

parameter	**symbol**	**connecting**	**cyclic**
start node	f_a	✓	✓
end node	f_b	✓	✓
turn angle	θ	✓	
strafe angle	ϕ		✓
speed	s		✓
gait type	g		✓

The parameters f_a, f_b determine where M_{i+1} connects from and to $\mathbf{G}_i$, θ gives the angle by which the character should turn during a transition motion, and ϕ, s, and g give the strafing angle, speed, and gait type of the cycle portion of M_{i+1}. Each such possible gait g is specified as in chapter 3 by a set C of *contact intervals* $(foot_1, start_1, end_1), \cdots, (foot_n, start_n, end_n)$, specifying that $foot_j$ is in contact with the ground from time $start_j$ to end_i. Specifying these contact intervals is relatively easy and generally takes only a few minutes. Further details on the formulation of the spacetime constraints optimization are given in appendix C.

Motion Initialization

Unfortunately, spacetime constraints optimizations such as the ones employed for generating the individual motions can be relatively sensitive to their initialization, $\widetilde{M}_{i+1}$. Depending on the quality of its initialization an optimization may converge to a good result, converge to a poor local minimum, or fail to converge at all. Although this can be mitigated by using the basin-CMA optimization (section 3.3.2), running this optimization many times to generate a vocabulary of diverse motion clips is prohibitively show. Since there is no assumption of any example data from which a good initialization might be formed this presents a difficulty. The key is noting that $\mathbf{G}_i$ is constructed entirely of physically valid motions seamlessly stitched together, and thus $\mathbf{G}_i$ in some sense represents a database of

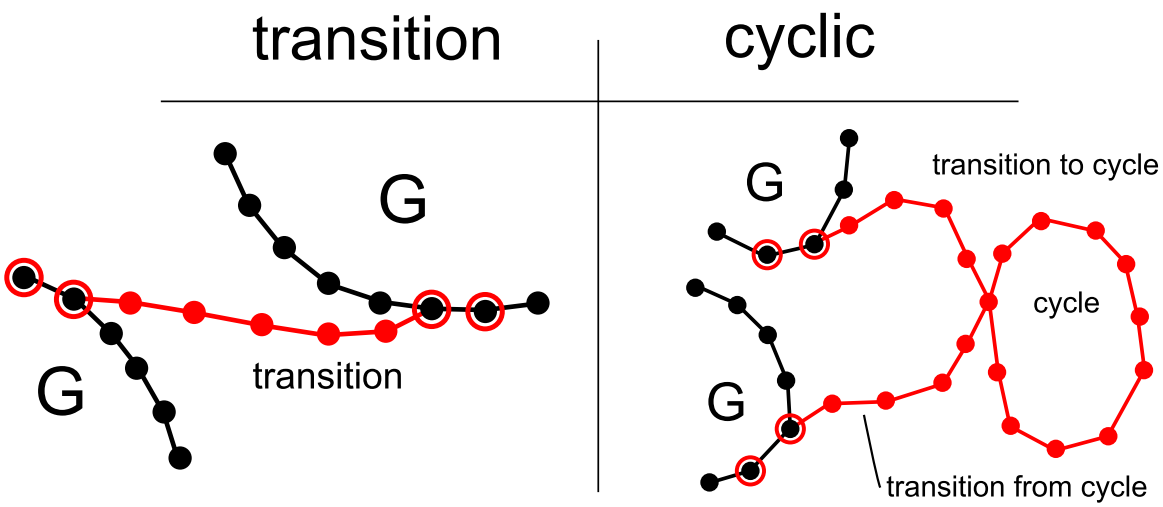


Figure 5.3: The two optimization templates employed to create the controllers demonstrated in this thesis. The transition template adds a transition between two existing frames in $\mathbf{G}$. The cyclic template creates a new cyclic motion M_c, then creates a transition motion from $\mathbf{G}$ to M_c and from M_c to $\mathbf{G}$. Note that the first and last pair of frames in each transition motion must match existing frames in $\mathbf{G}$.

'good' motions which can be used to create $\widetilde{M}_{i+1}$.

Since there are in general many possible initialization motions that can be generated from $\mathbf{G}_i$, it is necessary to have a means by which the quality of one candidate initialization, $\widetilde{M}_a$ may be judged against another $\widetilde{M}_b$. This is achieved by defining a *quality function* $q(\widetilde{M})$ which associates each candidate motion with a real number representing how good an initialization it provides. A quality of 0 means that the optimization is virtually guaranteed to converge to a good result, while motions with large quality measures are likely to be more problematic. In practice there is a simple choice for q works well: the sum of the distance between each frame in $\widetilde{M}$ and the nearest frame in $\mathbf{G}_i$:

$$q(\widetilde{M}) = \sum_j \min_k d\big(\widetilde{M}(f_j), \mathbf{G}_i(f_k)\big) \tag{5.3}$$

where $d\big(M(f_i), \mathbf{G}(f_j)\big)$ is defined according to the metric used in [37].

For each potential optimization, defined by an optimization template S_j and a setting of its parameters σ_j, a search is made through $\mathbf{G}_i$ to find a motion which best serves

as an initialization for an optimization with parameters in σ_j. The end result will be a motion which correctly satisfies the f_a, f_b, θ, and g parameters in σ_j, but which may not satisfy the ϕ or s parameters (see table 5.2.2) nor be physically realistic until the final spacetime constraints optimization is solved. This search for the best initialization motion in $\mathbf{G}$ is performed slightly differently depending on if a cyclic or a transition motion is being created.

For cyclic motions, all cycles in $\mathbf{G}_i$ up to 1.5 seconds in length are searched over. Each such cycle defines a motion M_c, but the timing of this motion may not match the that of the motion to be generated. This is the case when S_i and σ_i specify a set of foot contact intervals C_i which the resulting gait should follow. This timing mismatch is addressed by warping M_c so that its foot contact timings match those desired in the new motion. Let the foot contact intervals implicit in M_c be C_c. First, any cycles for which a foot has a different number of contact intervals in C_c than in C are discarded. For each remaining cycle, a temporal alignment is calculated between C_c and C independently for each foot by pairing off the contact intervals for that foot in order. A per-foot *time warp* function $t_c = T_{foot_j}(t)$ is then defined such that $T_{foot_j}(start_j) = start_{c,j}$ and $T_{foot_j}(end_j) = end_{c,j}$ for each $foot_j, start_j, end_j$ in C_i and $foot_j, start_{c,j}, end_{c,j}$ in C_c which match the given foot. Times in between the interval endpoints are linearly interpolated. Since T_{foot_j} perfectly aligns the contact timings of $foot_j$, an initialization motion $\widetilde{M}_i$ is created by blending between these different warps according to

$$\widetilde{M}_i(t)_d = \frac{\sum_{foot_j} w(foot_j, d) M_c(T_{foot_j})_d}{\sum_{foot_j} w(foot_j, d)}$$

where d ranges over the DOFs for the character and $w(foot_j, d)$ is 1 if DOF d is on a path from $foot_j$ to the root and 0.01 otherwise. This ensures that the contact timings of each foot in $\widetilde{M}_i$ match those in C_i and that the other degrees of freedom are reasonably interpolated. Finally, in the case that S_i has a parameter corresponding to the character's velocity, the motion of the root of the character in $\widetilde{M}_i$ is altered so that it achieves the desired velocity.

For transition motions a similar operation is performed. For a transition motion starting

at graph node f_a and ending at f_b, two sets of paths in $\mathbf{G}_i$ are searched over. One set, paths_a, consists of all paths starting from f_a (and ending anywhere) and the other set, paths_b, consists of all paths ending at f_b (but starting anywhere). For each possible combination of one motion, M_a, from paths_a and another motion, M_b, from paths_b a linear blend is computed to generate a motion which starts out identical to a M_a and which ends identically to M_b (interpolating in between). The motion resulting from this blend which results in the best score according to equation 5.3 is then chosen.

In the case that S_i parameterizes over a turn angle θ, this resulting clip is circularly warped so that it achieves the desired turn. The final surrogate motion, $\widetilde{M}_{i+1}$, is then calculated by performing a more expensive but higher quality blend of these two motions (M_a and M_b) using a registration curve as described in [35].

Graph Construction

The construction of a controller proceeds incrementally by augmenting a motion graph $\mathbf{G}_i$ with new motions generated from a spacetime constraints optimization. This requires determining which motion to add at each iteration. Since the goal is to arrive at a controller which minimizes equation 5.2, a hypothetical approach might search directly over the parameter spaces for each optimization template for the motion which when added to $\mathbf{G}_i$ gives the lowest value of $cost(\mathbf{G}_{i+1})$. Unfortunately correctly evaluating $cost(\mathbf{G}_{i+1})$ requires M_{i+1} which involves solving a spacetime constraints optimization – an extremely expensive computation and one which may often fail to converge to any solution at all. This makes it infeasible to directly use $cost(\mathbf{G}_{i+1})$ as the objective function for determining how $\mathbf{G}_i$ should be augmented.

Since actually calculating $cost(\mathbf{G}_{i+1})$ is expensive, I instead efficiently approximate it with a pair of *surrogate functions*: $\widetilde{cost}$ and $\widetilde{P}$. The function $\widetilde{cost}(\mathbf{G}_i, \sigma_{i+1})$ estimates $cost(\mathbf{G}_{i+1})$ under the assumption that the optimization $S_{i+1}(\widetilde{M}_{i+1}, \sigma_{i+1})$ is used to create M_{i+1} and that this optimization succeeds. $\widetilde{P}(\mathbf{G}_i, \sigma_{i+1})$ gives an estimate of the probability with which this optimization can be expected to succeed. The next optimization to perform is then found by searching for the optimization parameters σ^*_{i+1} which minimize the expected

value of the decrease in the cost of $\mathbf{G}_{i+1}$:

$$(1 - \widetilde{P}(\mathbf{G}_i, \sigma^*_{i+1})) \cdot cost(\mathbf{G}_i) + \widetilde{P}(\mathbf{G}_i, \sigma^*_{i+1})) \cdot \widetilde{cost}(\mathbf{G}_i, \sigma^*_{i+1}) \tag{5.4}$$

Once this search is completed, only a single spacetime constraints problem, $S_{i+1}(\widetilde{M}_{i+1}, \sigma^*_{i+1})$, is solved to create M_{i+1} and thereby $\mathbf{G}_{i+1}$.

The surrogate function, $\widetilde{cost}(\mathbf{G}_i, \sigma_{i+1})$, estimating the cost of a new graph generated by augmenting $\mathbf{G}_i$ with the result of an optimization performed with σ_{i+1} is defined by simply using $\widetilde{M}_{i+1}$ as a stand-in for M_{i+1}; creating $\widetilde{\mathbf{G}}_{i+1}$ by adding $\widetilde{M}_{i+1}$ to $\mathbf{G}_i$ and defining $\widetilde{cost}(\mathbf{G}_i, \sigma_{i+1}) = cost(\widetilde{\mathbf{G}}_{i+1})$. This avoids performing a spacetime constraints optimization, but has the caveat that it still requires estimates for the biomechanical costs in equation 5.1 associated with each frame in $\widetilde{M}_{i+1}$. This is achieved by interpolating the costs associated with each frame in $\mathbf{G}_i$ using:

$$\frac{\sum_{j=1}^{k} cost(M(f_j)) d(M(f_i)', M(f_j))^{-2}}{\sum_{j=1}^{k} d(M(f_i)', M(f_j))^{-2}}$$

where $d(M(f_i)', M(f_j))$ is measured according to the metric in [37].

The definition of $\widetilde{P}(\mathbf{G}_i, \sigma_{i+1})$ is necessarily more *ad hoc* as the probability that a given optimization will converge depends in complicated ways on the specifics of the optimization and the particular nonlinear optimizer used. In general there is no really accurate way of estimating this probability other than to simply run the optimization and see if it succeeds. Nevertheless, as a general principle it is expected that optimizations for which $\widetilde{M}_{i+1}$ has an initialization quality near zero will be likely to succeed, as well as optimizations for which σ_{i+1} is similar to an optimization which has previously succeeded. Similarly, optimizations for which σ_{i+1} is similar to those for a previously failed optimization will be likewise expected to fail, although this probability can of course be reduced at some later point by the the addition of other motions allowing a new higher quality initialization.

These intuitive properties of the probability with which an optimization is expected to succeed depend on a measurement of the similarity of two sets of optimization parameters. This notion of two sets of optimization parameters σ_a and σ_b being 'similar' can

be formalized by defining a distance metric $d(\sigma_a, \sigma_b)$ calculated as the square root of a sum per-parameter similarity measures as given below (refer to table 5.2.2 for parameter definitions):

parameter	**distance**
f_a	$d(\mathbf{G}_i(f_1), \mathbf{G}_i(f_2))^2$
f_b	$d(\mathbf{G}_i(f_1), \mathbf{G}_i(f_2))^2$
θ	$\sin(\theta_1)\sin(\theta_2) + \cos(\theta_1)\cos(\theta_2)$
ϕ	$\sin(\phi_1)\sin(\phi_2) + \cos(\phi_1)\cos(\phi_2)$
s	$(s_2 - s_1)^2$
g	δ_{g_1,g_2}

where $d(\mathbf{G}_i(f_1), \mathbf{G}_i(f_2))^2$ is again measured according to the metric in [37].

It remains to combine both the parameter similarity and initialization quality into a single number between 0 and 1 representing $\widetilde{P}(\mathbf{G}_i, \sigma_{i+1})$. This is done by considering each previous successful optimization defined by $\sigma_{s,j}$ with initialization quality $q_{s,j}$, as well as each failed optimization defined by $\sigma_{f,j}$ with initialization quality $q_{f,j}$. Denoting the initialization quality of $\widetilde{M}_{i+1}$ with q, the computation of $\widetilde{P}(\mathbf{G}_i, \sigma_{i+1})$ is then defined as:

$$\frac{S_1 + S_2}{S_1 + S_2 + F_1 + F_2} \tag{5.5}$$

where

$$S_1 = \sum_{q_{s,j}<q} \frac{q_{s,j}}{q} d(\sigma_{s,j}, \sigma_{i+1})^{-2} \tag{5.6}$$

$$S_2 = \sum_{q_{s,j}>q} \frac{1}{d(\sigma_{s,j}, \sigma_{i+1})^2 + \log(\frac{q}{q_{s,j}})} \tag{5.7}$$

$$F_1 = \sum_{q_{f,j}>q} \frac{q}{q_{f,j}} d(\sigma_{f,j}, \sigma_{i+1})^{-2} \tag{5.8}$$

$$F_2 = \sum_{q_{f,j}<q} \frac{1}{d(\sigma_{f,j}, \sigma_{i+1})^2 + \log(\frac{q_{f,j}}{q})} \tag{5.9}$$

Although I have found this particular definition of $\widetilde{P}(\mathbf{G}_i, \sigma_{i+1})$ to work well, it is likely

that many other techniques would also work so long as they satisfy the previously described general properties of $\widetilde{P}(\mathbf{G}_i, \sigma_{i+1}) = 1$ when $q = 0$, $\widetilde{P}(\mathbf{G}_i, \sigma_{i+1}) = 1$ when $\sigma_{i+1} = \sigma_{s,j}$ for some j, and $\widetilde{P}(\mathbf{G}_i, \sigma_{i+1}) = 0$ when $\sigma_{i+1} = \sigma_{f,j}$ and $q \leq q_{f,j}$ for some j. As a visual aid, figure 5.7 shows plots of the quantities relevant to the construction of a motion graph during the creation of a graph for controlling the walking direction of a bipedal character.

Parameter Search

Having defined $\widetilde{cost}(\mathbf{G}_i, \sigma_{i+1})$ and $\widetilde{P}(\mathbf{G}_i, \sigma_{i+1})$, it only remains to give the specifics of how to search for the optimization parameters σ^*_{i+1} which minimize equation 5.4. Although in principle any optimization technique would suffice for this task, an efficient approach can be obtained by noting that although $\widetilde{cost}(\mathbf{G}_i, \sigma_{i+1})$ and $\widetilde{P}(\mathbf{G}_i, \sigma_{i+1})$ are significantly cheaper to compute than actually performing a spacetime constraints optimization, they are still relatively expensive (on the order of 0.2 to a few seconds). An efficient optimization method must therefore endeavor to reduce how many evaluations of the surrogate functions are actually required.

Because the the parameter spaces for each optimization template can be expected to be low-dimensional, I employ a method which utilizes all previous evaluations of equation 5.4 for the current $\mathbf{G}_i$ iteration. Specifically, given the previous evaluations of equation 5.4 $(\sigma_1, cost_1), \ldots, (\sigma_n, cost_n)$ such that $cost_j$ is the evaluation of equation 5.4 for the parameters σ_j, the next parameters σ_{n+1} to sample are chosen as those which minimize $\sum_j \frac{w_j \, cost_j}{w_j + 1}$ where $w_j = d(\sigma_{n+1}, \sigma_j)^{-2}$. This equation has the property that it is exactly equal to $cost_j$ when $\sigma_{n+1} = \sigma_j$, and generally decreases elsewhere, leading to a global exploration of the optimization parameter space.

Once this search process converges, a set of optimization parameters σ^*_{i+1} have been found which minimize equation 5.4, so a new motion graph can be constructed by using σ^*_{i+1} to create M_{i+1} and thereby $\mathbf{G}_{i+1}$, repeating the process until either a given size of $\mathbf{G}$ is attained or when the decrease of $cost(\mathbf{G}_{i+1})$ from $cost(\mathbf{G}_i)$ falls below some threshold. Pseudocode for this process appears in lines 4-15 of figure 5.2. The final motion graph defines a controller which can animate the input animal efficiently performing the given

tasks.

5.3 Results

To demonstrate the effectiveness of the automatic controller generation method I will illustrate a variety of different controller types over a range of different animal shapes. These animals are chosen to have widely varying morphologies: Two different simplified bipedal creatures both with 16 DOFs and weighing 10.6kg, a 65kg triped with a large back leg and two smaller front legs with 22 DOFs, and a 500kg horse-like quadruped with 29 DOFs. The skeletal structures for these animals are illustrated in figure 5.4.

The task models used to define the input to these controllers consist of both single- and multi-objective tasks. Each of these task models is created by combining one or multiple of four different sub-tasks: a direction following task, a torso angle matching task, a speed matching task, and a gait matching task, each parameterized by a single variable and with an associated cost function. The total task cost function for equation 5.1 is simply the sum of the individual cost functions:

task	**task parameter**	**cost**
direction following	θ_v	$\lvert\theta_v - \text{atan}(\frac{v_y}{v_x})\rvert$
torso orientation	θ_r	$\lvert\theta_r - \phi\rvert$
speed	s_t	$\lvert s_t - \lVert v\rVert\rvert$
gait type	g_t	$10 \cdot \delta_{g_t,g}$

Figure 5.5 shows some frames from a triped controller constructed for a task model parameterizing over both direction and gait type.

I will demonstrate the controllers resulting from four different combinations of these atomic tasks. Each controller is created to allow simultaneous control over all its constituent tasks, so for instance a direction+torso controller allows both the direction in which the animal is moving and the direction its torso is facing to be simultaneously controlled. In all cases the synthesized controller successfully performs the desired tasks without need to any manual per-task or animal tweaking:

demo	**task parameters**	**optimization parameters**
direction	θ_v	θ
direction + torso	θ_v, θ_r	θ, ϕ
direction + gait	θ_v, g_t	θ, g, f_a, f_b
speed	s_t	s, f_a, f_b

The learning times observed for these controllers range from 5-10 minutes for a biped direction controller to 8-10 hours for a horse torso angle plus direction controller. Successful controllers tend to require between 5-10 constituent motions for the single-parameter controllers to 15-30 clips for more complicated controllers. Image sequences illustrating animations created by user interaction with these controllers are illustrated in figures 5.8, 5.9, 5.11, 5.12, 5.13, 5.14, and 5.15.

Since part of the goal of my controller synthesis algorithm was not just to create controllers automatically, but to do so relatively efficiently, I evaluate the effectiveness of the process for iteratively constructing $\mathbf{G}$ by comparing it against a technique which attempts to sample the parameter space of each S_i as uniformly as possible. A graph of $cost(\mathbf{G}_i)$ at each iteration for my approach versus uniform sampling for three different task descriptions is shown in figure 5.6. For the two parameter task models, my approach achieves a better cost after its first few iterations than was achieved by uniform sampling even after over 20 iterations. The improvement for single parameter tasks is less dramatic since the state space is small enough to be sampled simply. Nevertheless the relative improvement of our approach is still substantial – after 20 iterations achieving an expected cost over five times lower than by uniform sampling.

As discussed in section 5.2.1 each controller requires not only a definition of the tasks for which it will be used, but also the dynamics for how the goals for these tasks are expected to change over time. My implementation does this independently for each task parameter by randomly choosing between keeping the parameter unaltered, switching it randomly to a new value, or changing it to a value randomly selected from a Gaussian distribution centered at the parameter's current value. Changing the relative probability with which these options occur impacts the type of controller created. For instance, controllers generated under large

and frequent changes to the task parameters focus on motions which allow the character to quickly change its state, while controllers which assume that the task parameters stay primarily constant excel at making sure that the character can eventually satisfy the task extremely accurately.

Additionally, setting the weight given to biomechanical versus task costs (equation 5.1) provides a simple and useful way to alter the nature of the generated controllers. Controllers with a low weight on biomechanical costs are more agile, but may appear unrealistic, while controllers emphasizing the biomechanical costs are more smooth and natural looking, but adapt more slowly to changes in the task parameters. I have found a that a wide range of weights give reasonable results, and typically weight the biomechanical costs at 1 to 0.1 relative to the the task costs.

As is the case with almost all methods which control a character using a Markov decision process, the method described here scales poorly with the number of task parameters used simultaneously. This is both because solving for $\widetilde{cost}$ is more expensive, and because larger motion graphs are usually required to perform well in such cases. This, combined with the expense of solving the optimization for each motion in the controller graph, limits the applicability of my approach to problems with two or perhaps three task parameters. A final limitation on my approach is its fundamental reliance on a powerful trajectory optimizer. Although the spacetime constraints implementation described in chapter 3 performs reasonably well at this task, it has trouble synthesizing motions for animals with more than 30-40 DOFs, particularly where said animal is executing a higher speed run or jog.

k

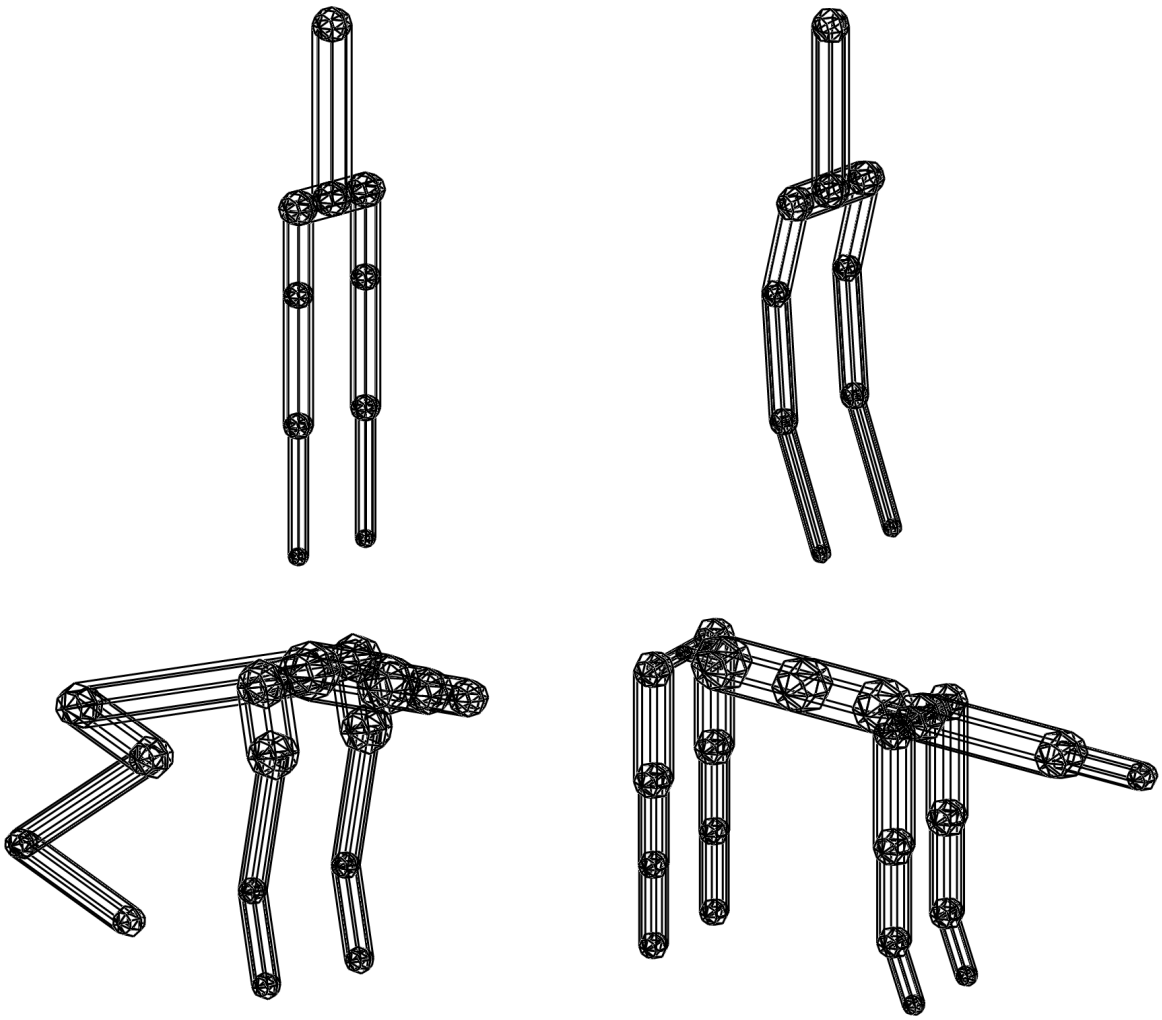


Figure 5.4: The skeletal structures for which the automatic controller generation will be illustrated. These skeletons consist of two different bipeds, a triped, and a quadruped as illustrated.

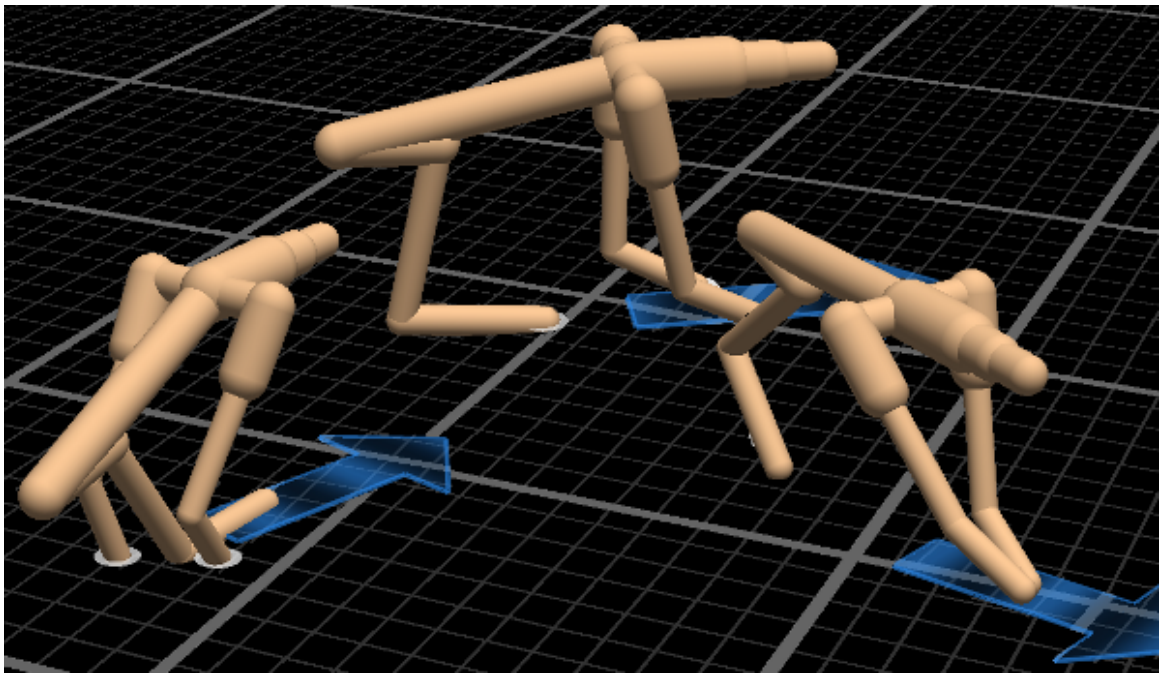


Figure 5.5: Frames of a triped direction and gait type controller in use.

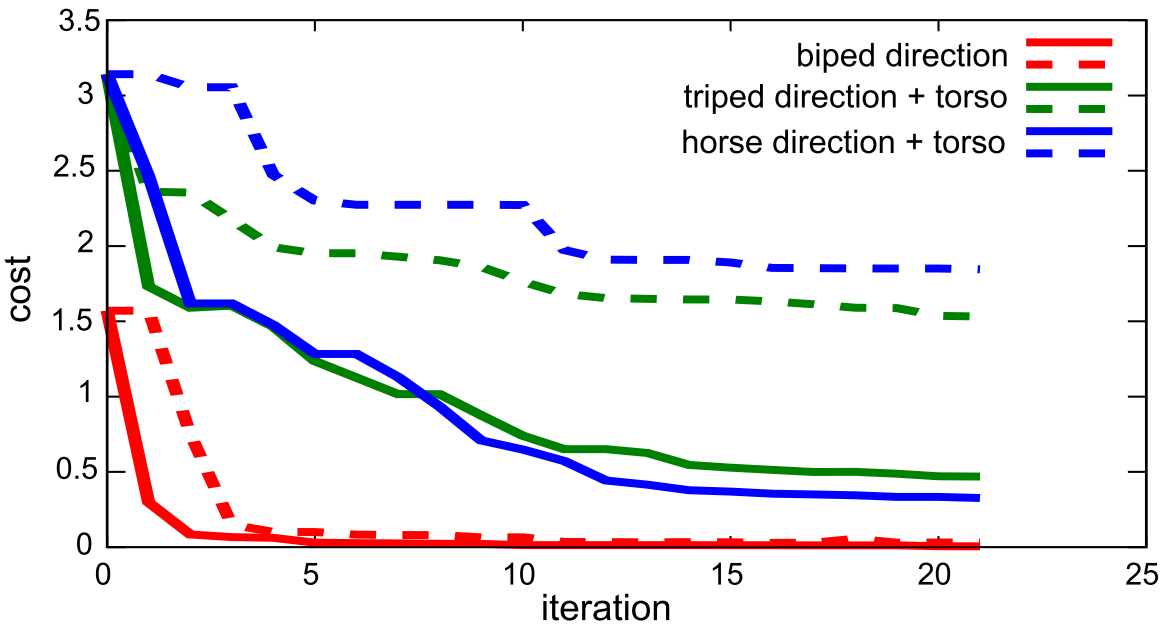


Figure 5.6: Solid lines plot $cost(\mathbf{G}_i)$ over 20 iterations of the controller generation algorithm for three different task models. For comparison the costs achieved by a uniform sampling approach for the same tasks are shown as dashed lines.

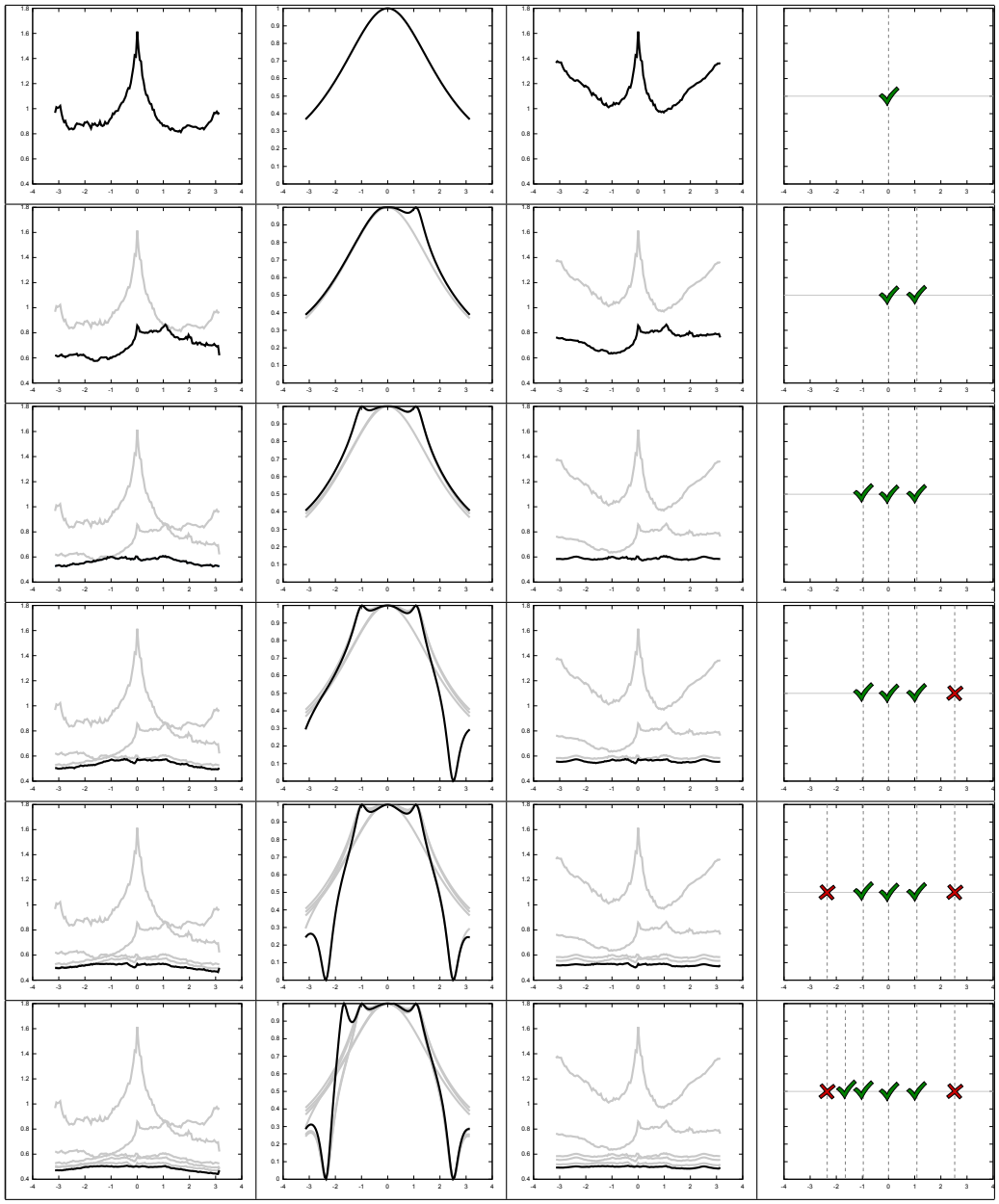


Figure 5.7: From top to bottom consecutive iterations are shown for adding new motions to a biped direction controller, plotted for $\mathbf{G}_0$ to $\mathbf{G}_5$ in the synthesis process. The motions are parameterized by the angle by which the character turns during the motion. From left to right within each row are shown 1) $\widetilde{cost}$ for adding an turn edge to a biped direction controller at each angle, 2) the same plot for $\widetilde{P}$, 3) the same plot for the value of equation 5.4, and 4) the angle at which each motion was synthesized and if the motion synthesis was successful.

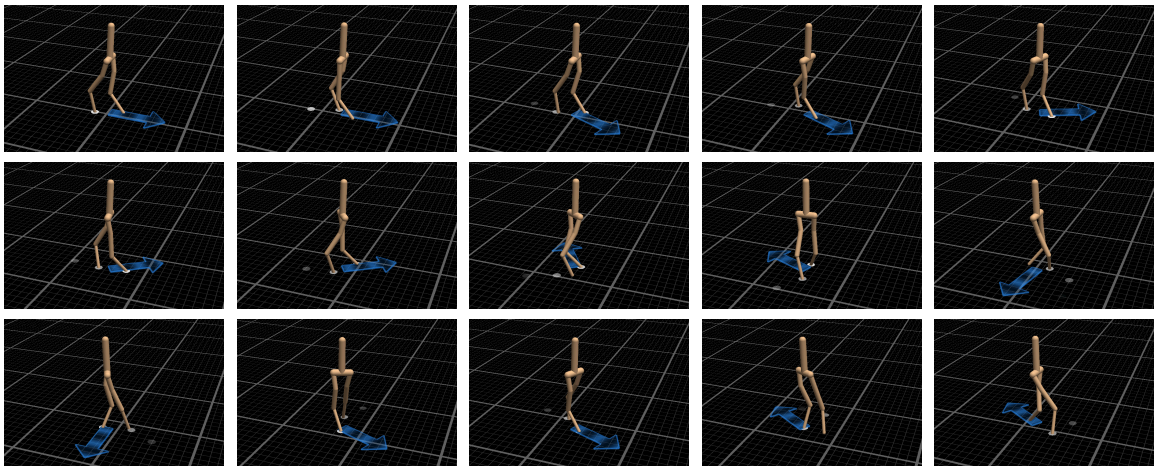


Figure 5.8: Selected frames from an animation recorded from a user's interaction with a controller synthesized for a simplified bipedal creature and allowing control over the direction in which the character should walk (blue arrow).

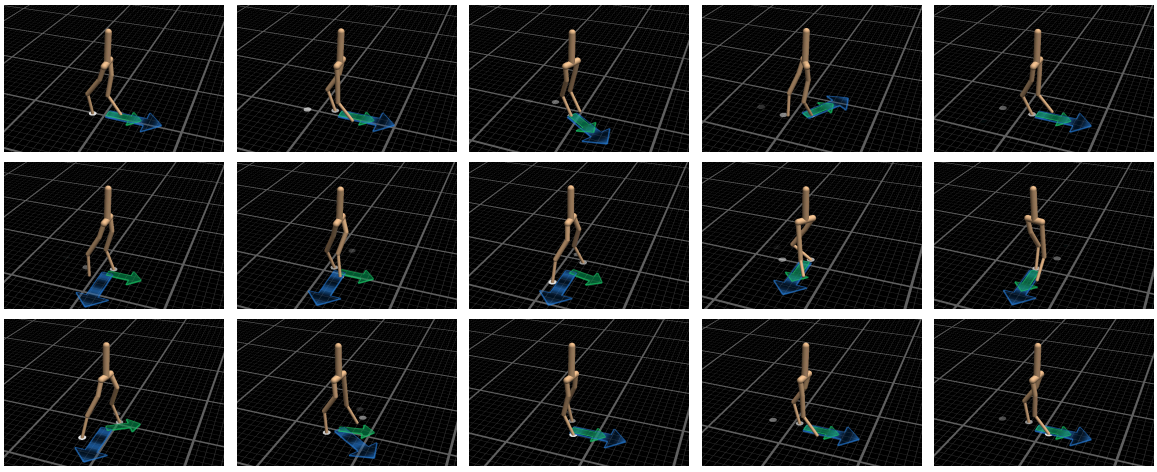


Figure 5.9: Selected frames from an animation recorded from a user's interaction a with controller synthesized for a simplified bipedal creature and allowing control over the direction in which the character should walk (blue arrow) and the direction in which the character should face (green arrow).

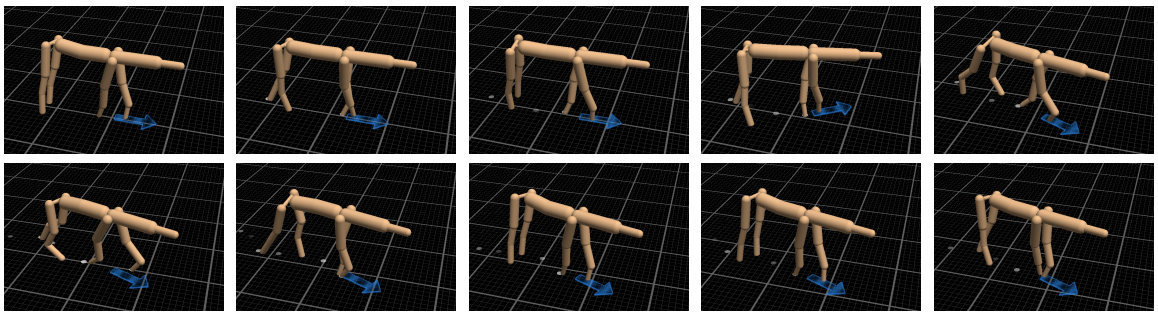


Figure 5.10: Selected frames from an animation recorded from a user's interaction with a controller synthesized for a horse-like quadrupedal creature and allowing control over the direction in which the character should walk (blue arrow).

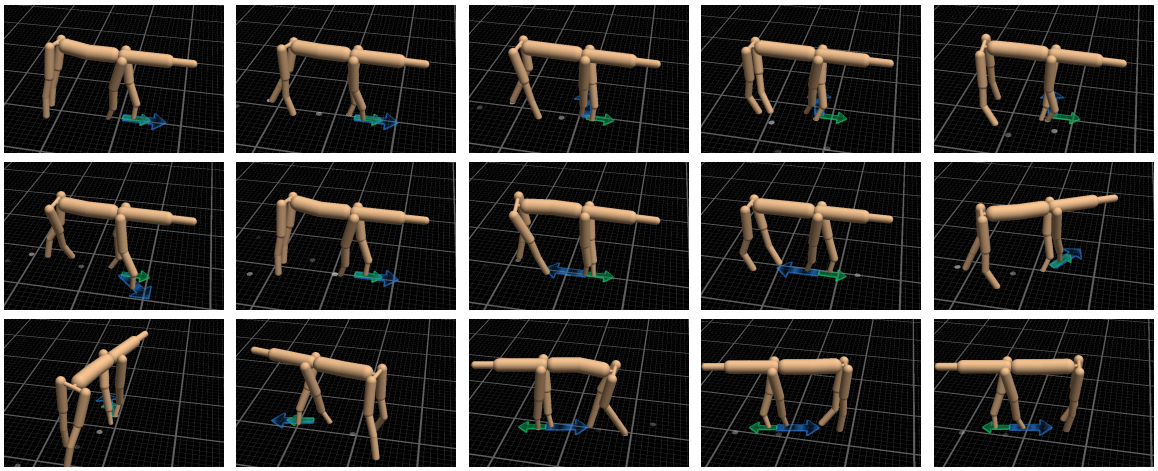


Figure 5.11: Selected frames from an animation recorded from a user's interaction with a controller synthesized for a horse-like quadrupedal creature and allowing control over the direction in which the character should walk (blue arrow) and the direction in which the character should face (green arrow).

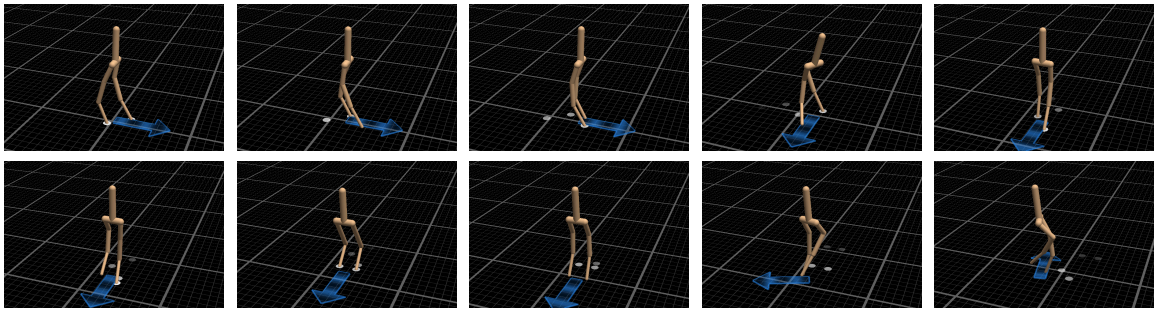


Figure 5.12: Selected frames from an animation recorded from a user's interaction with a controller synthesized for a simple bipedal creature with a rather odd leg configuration and allowing control over the direction in which the character should walk (blue arrow) as well as whether the character should walk or hop.

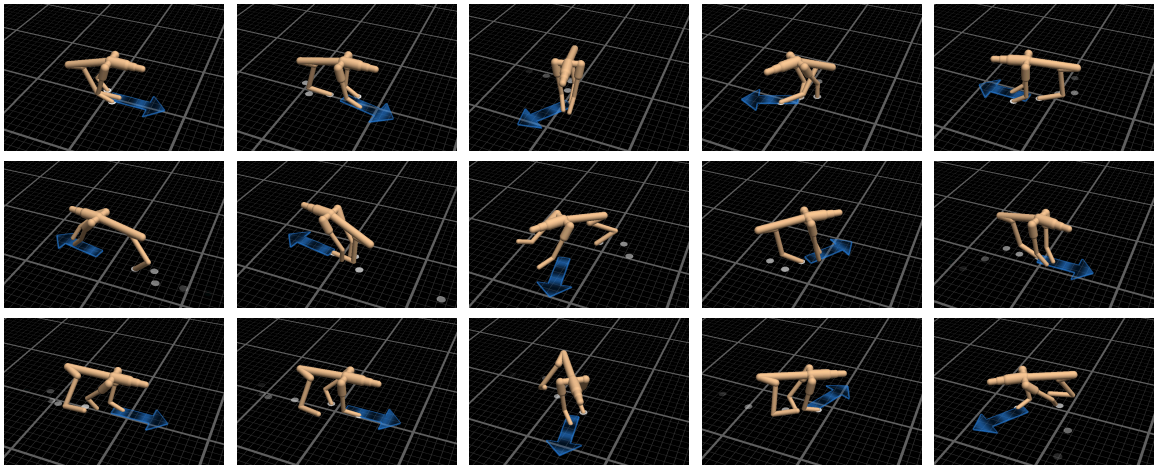


Figure 5.13: Selected frames from an animation recorded from a user's interaction with a controller synthesized for a tripedal creature and allowing control over the direction in which the character should walk (blue arrow) as well as the style of gait (walk, run, or bipedal walk).

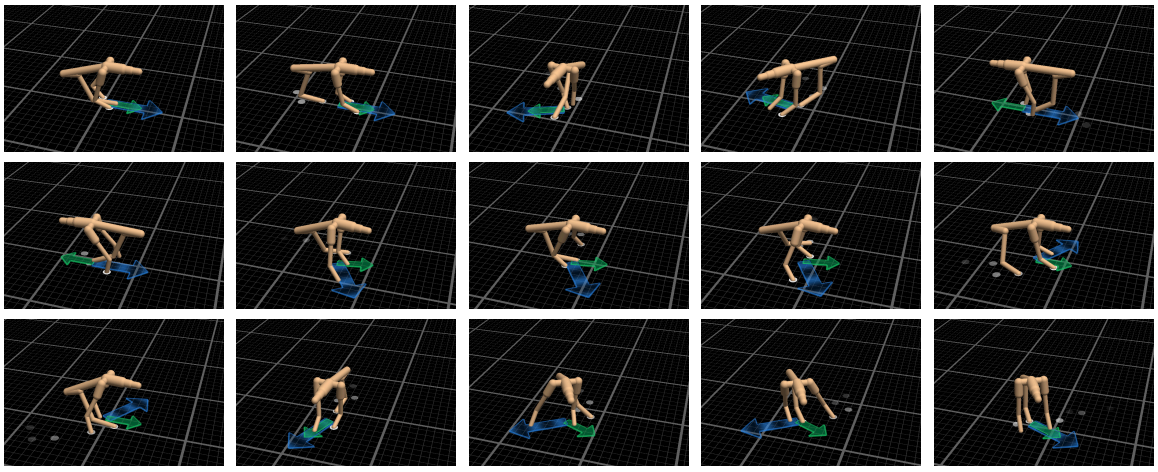


Figure 5.14: Selected frames from an animation recorded from a user's interaction with a controller synthesized for a tripedal creature and allowing control over the direction in which the character should walk (blue arrow) and the direction in which the character should face (green arrow).

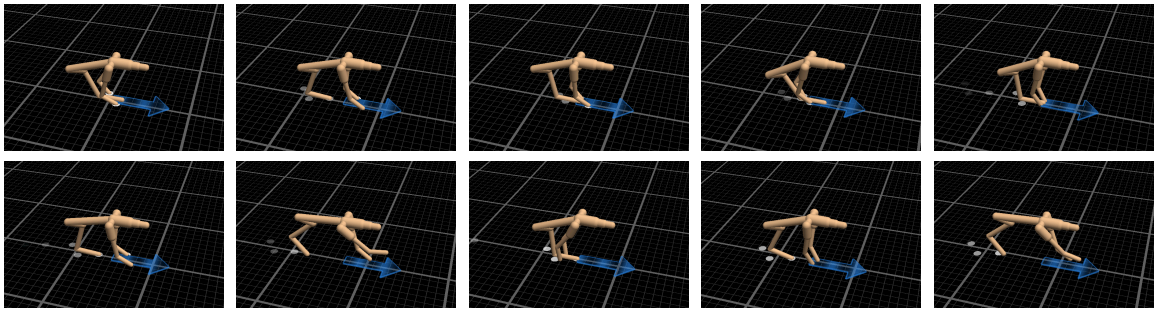


Figure 5.15: Selected frames from an animation recorded from a user's interaction with a controller synthesized for a tripedal creature and allowing control over the speed at which the character should walk/run. The choice of weather to use a walking or a running gait at each speed is determined automatically by the minimization of the expected cost of running the controller.

Chapter 6

CONCLUSION

In this thesis I have described computational techniques capable of automatically synthesizing locomotion for an input animal. These techniques do not rely on any data for the animal in order to perform this synthesis, but rather determine how the animal should move by employing the laws of physics combined with biomechanical principles. These approaches are not limited to any particular type of animal and can be applied equally to bipeds, quadrupeds, and fictional creatures such as monopeds, tripeds, and pentapeds. In particular, I have shown techniques addressing three sub-problems core to the synthesis of animal locomotion:

1. Synthesis of gaits from an input skeleton.

2. Improvement of the synthesized gaits by employing a database of motions for animals of various shapes and sizes.

3. Synthesis of real-time kinematic locomotion controllers from an input skeleton and a description of the task(s) for which the controller will be used.

All of the synthesis methods described in this thesis involve formulating the motion synthesis problem as a constrained optimization. This formulation ensures that the resulting motions exactly obey the laws of physics and, in addition to this, minimize the effort required by the animal to perform the motion. This constrained optimization approach allows a consistent formulation between all of the components to this thesis, and each of the approaches is based on what is essentially the same definition of what counts as an 'optimal motion'. This mathematical uniformity between the solutions to the three different sub-problems also serves to facilitate their interoperability, so that future advances in one of these sub-problems will likely lead directly to an improvement in the other as well.

6.1 Contributions

In this thesis I have covered several aspects related to automatic locomotion synthesis for animals. In doing so, I have introduced several novel contributions relating to both character animation and to optimization:

***De novo* animal gait synthesis** To the best of my knowledge the method presented here was the first to demonstrate the ability to synthesize a physically-realistic gait for an animal without requiring any pre-captured motion data. This was achieved by a spacetime constraints optimization tailored directly to animal gait synthesis.

Optimization over gait type I have presented a method by which the style of gait employed by an animal may be optimized over. This allows, for instance, the automatic determination of whether a walk or a run would be more efficient at a particular speed. This optimization was performed by a novel hybridization of the sampling-based approach of the covariance matrix adaptation evolution strategy with a derivative-based constrained optimizer and can significantly outperform existing approaches to global constrained optimization.

Optimization over morphology I further show how the spacetime constraints optimization used to synthesize animal gaits can be extended to co-optimize over the animal's shape as well. To the best of my knowledge the method presented in this thesis is the first to attempt this for general animals and which is applicable to animals with a shape similar to those found in nature. The reshaping of an animal's shape can be used to refine an original design for an animal so that the animal can perform a particular task more efficiently, and often leads to smoother and more natural appearing motions.

Joint inverse optimization In order to increase the realism of the gaits synthesized for an animal I have described an approach based on joint inverse optimization. In this approach a database of 'ground truth' motions is extracted from videos of animal locomotion, and the motions in this database are used to determine the appropriate

definition of 'effort' so that the low-effort motions under this definition are as realistic as possible. This refined definition of effort can then be used to synthesize motions for new animals for which there is no motion data available.

***De novo* animal locomotion controller synthesis** In addition to the synthesis of gait cycles, I have described an approach for automatically synthesizing a kinematic controller given an animal and a description of the tasks for which the controller will be used. The controller can be directed by a user in real-time, and although it is kinematic the synthesized motions are physically realistic. This controller synthesis method also attempts to minimize the computational time required to generate a controller.

6.2 Future Work

The motion of animals is a large topic, so there remain many potential avenues where future research might extend the ideas I have presented in this thesis. In the following subsections I will briefly describe a few of what I believe to be the most promising and interesting directions for future research.

6.2.1 Optimization Improvements

Many approaches to character animation, including those described in this thesis, fundamentally rely on the ability to solve relatively difficult optimization problems. The success and range of applicability of these approaches is thus dependent on the abilities of the underlying optimizer. Even though the optimizations I have used in this thesis work relatively well compared to existing methods, there is still a great deal of room for future innovation.

The core optimizations underlying all of the motion synthesis approaches I have presented in this thesis are formulated as spacetime constraints optimizations. These formulations involve phrasing the motion synthesis problem as a nonlinear constrained optimization, and this constrained optimization is then solved with an off-the-shelf optimization package (SNOPT [19] in the examples in this thesis). Although the spacetime constraints formulations I have used are relatively efficient and robust compared to preexisting methods, the

speed and robustness with which these optimizations can be solved is still a major difficulty in more complex applications.

Although the optimizations arising in character animation are known for being relatively difficult to solve, there is some hope that efficient optimizers can be created. In particular, current approaches rely on the success off-the-shelf optimization packages which are not specifically tailored to character animation. Many of the optimizations arising in character animation share some common structural characteristics, and it is likely that these common structures can be exploited to create optimizers better suited to character motion synthesis. Some recent research such as [58]and [74] have begun to move in this direction by reconsidering how contacts should be treated (thereby allowing contacts to be solved for more efficiently), but there are likely many other advances to be discovered in this area.

In addition to the improvement of the core spacetime constraints optimizations discussed in this thesis, many of the other optimizations presented as extensions of these spacetime constraints approaches also have room for future improvement. For instance, in section 3.4 I presented an approach wherein an animal's form could be co-optimized along with its motion to efficiently perform a given task. The adaptation of the animal's form, however, was constrained to have the same topology as the original animal. A more sophisticated optimization method could potentially avoid this restriction, and determine, for instance, the optimal number of legs for an animal to have.

In chapter 4 I introduced an approach for improving the realism of the motions synthesized for an animal by referencing a database of ground-truth motions for similar animals. One component to this approach involves fitting a physically realistic motion so that it best matches a motion seen in a reference video. Although the optimization I have presented to achieve this is have proved to be relatively robust, it is still somewhat limited in its application. In particular it relies on a series of points tracked from a side-on video of the animal moving at a constant speed with a fixed camera center. In practice, however, it turns out that many videos don't match this relatively specific form. A more thorough integration of the computer vision and optimization aspects of this optimization could hopefully eventually allow accurate 3D motions to be automatically extracted from an uncategorized set of videos of an animal including various different kinds of motions and camera behaviors.

Such a technique would have the additional benefit that it would facilitate in improving the realism of non-gait motions.

The approach described in chapter 4 also relies on solving a number of inverse optimization problems. The approach I have presented to do so is a somewhat brute-force method, and is relatively computationally expensive. Although there is at least one existing approach [48] which attempts to solve this sort of nonlinear inverse optimization more efficiently, it solves a slightly different optimization problem and I have not found it to perform well in the context of the inverse optimization problems in this thesis. Nevertheless, nonlinear inverse optimization problems have received relatively little attention so far within the optimization community, so there are likely much more robust and efficient approaches yet to be discovered. Such advances in techniques for inverse optimization would greatly aid in more complex analyses of animal motions.

Finally, chapter 5 involved an optimization for constructing a motion controller from a sequence of carefully chosen spacetime constraints optimizations. Within this context, the most exciting future possibility is to move beyond pure graph-based kinematic controllers. In particular, both dynamic controllers and some recent approaches to kinematic motion controllers such as [46] and [47] allow the character to deviate from the motions from which the controller is constructed. This deviation can happed in response to physical disturbances or user commands, and is crucial to achieving responsive and natural motions. The automatic synthesis of such motion controllers is an unaddressed problem which has great potential for automatically creating believable animations in real-time contexts.

6.2.2 More Complete Animal and Motion Models

In addition to the extensions and improvements to the optimizations employed in this thesis, there are many possibilities for future research to expand from the work presented here to more complete considerations of both the biology of individual animals and the diversity of different animals and their behaviors. These avenues for future research could potentially greatly increase the accuracy of the generated motions as well as the range of the applicability of the techniques used to generate them.

One of the most obvious extensions of the techniques presented here is to increase the fidelity of the biomechanical model used to model each animal. In particular, the methods used in this thesis do not include models of individual muscles, and use a relatively basic model of the passive actuation characteristics of an animal's tendons and ligaments. Also neglected are issues arising from the approximation of an animal as a collection of connected rigid bodies rather something consisting in part of soft and deformable tissues. Along a somewhat different vein, the spacetime constraints optimizations employed in the approaches presented here assume that there are no uncertainties in how an animal can actuate its joints and that the environment is known to exact accuracy. Relaxing these assumptions, and potentially accounting for effects such as a non-zero response time to any unexpected occurrences would likely improve the realism of the resulting animations, in addition to potentially being of biological interest.

There are also a great many opportunities for capturing non-locomotion behaviors. Although techniques exist to model swimming and flying behaviors, there are no approaches which are able to model, for instance, transitions from walking to flying. Similarly, the synthesis of climbing and swinging motions remains largely unaddressed, as does the synthesis of non-legged locomotion such as that of snakes.

A final aspect for fruitful future work is the synthesis of non-locomotion behaviors. Although locomotion forms the most obvious type of terrestrial animal motion, non-locomotion behaviors are extremely common. The range of such behaviors, including leaping, lying down, digging, and grabbing to mention just a few is certainly very broad and relatively hard to completely classify. Accurately synthesizing this broad range of non-locomotion motions, however, is crucial for creating fully believable virtual animals. I expect future advances will greatly increase the capability of motion synthesizers to include a wide range of behaviors, giving rise to relatively full-features models of animal motion.

BIBLIOGRAPHY

[1] R. M. Alexander. *Optima for Animals*. Princeton, 1996.

[2] Okan Arikan and David A. Forsyth. Interactive motion generation from examples. *ACM Transactions on Graphics (ACM SIGGRAPH 2002)*, 21(3):483–490, 2002.

[3] Aristotle. *De Motu Animalium*. C.340BC.

[4] Joel Auslander, Alex Fukunaga, Hadi Partovi, Jon Christensen, Lloyd Hsu, Peter Reiss, Andrew Shuman, Joe Marks, and J. Thomas Ngo. Further experiences with controller-based automatic motion synthesis for articulated figures. *ACM Transactions on Graphics*, 14(4):311–336, October 1995.

[5] J E Bertram and A Ruina. Multiple walking speed-frequency relations are predicted by constrained optimization. *J Theor Biol*, 209(4):445–53, 2001.

[6] Giovanni Alfonso Borelli. *De motu animalium [microform] / Io. Alphonsi Borelli ; opus posthumum*. Ex typographia Angeli Bernabo, Romae :, 1680.

[7] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[8] Armin Bruderlin and Lance Williams. Motion signal processing. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, pages 97–104, August 1995.

[9] Liming Zhao Cheng Ren and Alla Safonova. Human motion synthesis with optimization-based graphs. 29(2), 2010.

[10] Min Gyu Choi, Jehee Lee, and Sung Yong Shin. Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics*, 22(2):182–203, April 2003.

[11] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Robust task-based control policies for physics-based characters. *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*, 28(5):Article 170, 2009.

[12] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Generalized biped walking control. *ACM Transctions on Graphics*, 29(4):Article 130, 2010.

[13] Stelian Coros, Philippe Beaudoin, KangKang Yin, and Michiel van de Panne. Synthesis of constrained walking skills. *ACM Transactions on Graphics*, 27(5):113:1–113:9, 2008.

[14] Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel van de Panne. Locomotion skills for simulated quadrupeds. *ACM Transactions on Graphics*, 30(4):Article TBD, 2011.

[15] Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. Feature-Based Locomotion Controllers. *ACM Transactions on Graphics*, 29(3), 2010.

[16] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. In *Proceedings of ACM SIGGRAPH 2001*, Annual Conference Series, pages 251–260, 2001.

[17] Anthony C. Fang and Nancy S. Pollard. Efficient synthesis of physically valid human motion. *ACM Trans. Graph.*, 22(3):417–426, 2003.

[18] R. Featherstone. *Rigid Body Dynamics Algorithms*. Kluwer international series in engineering and computer science: Robotics. Springer, 2008.

[19] Philip E. Gill, Walter Murray, and Michael A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005.

[20] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, pages 9–20, July 1998.

[21] Brian Guenter. Efficient symbolic differentiation for graphics applications. *ACM Trans. Graph.*, 26(3):108, 2007.

[22] K. Häkkinen and K. L. Keskinen. Muscle cross-sectional area and voluntary force production characteristics in elite strength- and endurance-trained athletes and sprinters. *European Journal of Applied Physiology*, pages 215–220, Apr 2006.

[23] N. Hansen and S. Kern. Evaluating the CMA evolution strategy on multimodal test functions. In X. Yao et al., editors, *Parallel Problem Solving from Nature PPSN VIII*, volume 3242 of *LNCS*, pages 282–291. Springer, 2004.

[24] Nikolaus Hansen, Nikolaus Hansen, Andreas Ostermeier, and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. pages 312–317. Morgan Kaufmann, 1996.

[25] Samuel Haughton. *Principles of animal mechanics*. London: Longmans, Green and Co, 1873.

[26] Rachel Heck and Michael Gleicher. Parametric motion graphs. *Proceedings of Symposium on Interactive 3D Graphics and Games (I3D) 2007*, April 2007.

[27] Chris Hecker, Bernd Raabe, Ryan W. Enslow, John DeWeese, Jordan Maynard, and Kees van Prooijen. Real-time motion retargeting to highly varied user-created morphologies. *ACM Trans. Graph.*, 27(3):1–11, 2008.

[28] Jessica K. Hodgins and Nancy S. Pollard. Adapting simulated behaviors for new characters. In *Proceedings of SIGGRAPH 97*, pages 153–162, August 1997.

[29] Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. Animating human athletics. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, pages 71–78, 1995.

[30] Holmes. The Dynamics of Legged Locomotion: Models, Analyses, and Challenges. *Dynamics*, 48(2):207–304.

[31] Takeo Igarashi, Tomer Moscovich, and John Hughes. Spatial keyframing for performance-driven animation. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 107–116, July 2005.

[32] Leslie Ikemoto, Okan Arikan, and David Forsyth. Generalizing motion edits with gaussian processes. *ACM Trans. Graph.*, 28(1):1:1–1:12, February 2009.

[33] Eakta Jain, Yaser Sheikh, and Jessica Hodgins. Leveraging the talent of hand animators to create three-dimensional animation. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 93–102, New York, NY, USA, 2009. ACM.

[34] Sumit Jain and C. Karen Liu. Controlling physics-based characters using soft contacts. *ACM Trans. Graph. (SIGGRAPH Asia)*, 30:163:1–163:10, December 2011.

[35] Lucas Kovar and Michael Gleicher. Flexible automatic motion blending with registration curves. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '03, pages 214–224, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[36] Lucas Kovar and Michael Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph.*, 23:559–568, August 2004.

[37] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM Transactions on Graphics*, 21(3):473–482, July 2002.

[38] P. G. Kry, L. Reveret, F. Faure, and M.-P. Cani. Modal locomotion: Animating virtual characters with natural vibrations. *Computer Graphics Forum*, 2009.

[39] Joseph Laszlo, Michiel van de Panne, and Eugene Fiume. Interactive control for physically-based animation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 201–208, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[40] Manfred Lau and James J. Kuffner. Behavior planning for character animation. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 271–280, July 2005.

[41] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics*, 21(3):491–500, July 2002.

[42] Jehee Lee and Kang Hoon Lee. Precomputing avatar behavior from human motion data. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 79–87, July 2004.

[43] Seong Jae Lee and Zoran Popović. Learning behavior styles with inverse reinforcement learning. *ACM Trans. Graph.*, 29(4):1–7, 2010.

[44] Sung-Hee Lee, Eftychios Sifakis, and Demetri Terzopoulos. Comprehensive biomechanical modeling and simulation of the upper body. *ACM Trans. Graph.*, 28:99:1–99:17, September 2009.

[45] Yongjoon Lee, Seong Jae Lee, and Zoran Popović. Compact character controllers. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers*, pages 1–8, New York, NY, USA, 2009. ACM.

[46] Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. Motion fields for interactive character locomotion. *ACM Trans. Graph.*, 29(6):138:1–138:8, December 2010.

[47] Sergey Levine, Jack M. Wang, Alexis Haraux, Zoran Popović, and Vladlen Koltun. Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics*, 31(4):28, 2012.

[48] C. Karen Liu, Aaron Hertzmann, and Zoran Popović. Learning physics-based motion style with nonlinear inverse optimization. *ACM Trans. Graph.*, 24(3):1071–1081, 2005.

[49] Zicheng Liu, Steven J. Gortler, and Michael F. Cohen. F.: Hierarchical spacetime control of linked figures. In *In Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 35–42. ACM Press, 1994.

[50] M. Liverman. *The Animator's Motion Capture Guide: Organizing, Managing, and Editing*. Charles River Media Game Development. Charles River Media, 2004.

[51] Wan-Yen Lo and Matthias Zwicker. Real-time planning for parameterized human motion. In *2008 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 29–38, July 2008.

[52] James McCann and Nancy Pollard. Responsive characters from motion fragments. *ACM Transactions on Graphics (SIGGRAPH 2007)*, 26(3), July 2007.

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

[54] Thomas B. Moeslund and Erik Granum. A survey of computer vision-based human motion capture. *Comput. Vis. Image Underst.*, 81(3):231–268, March 2001.

[55] Thomas B. Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Comput. Vis. Image Underst.*, 104(2):90–126, November 2006.

[56] Igor Mordatch, Martin de Lasa, and Aaron Hertzmann. Robust Physics-Based Locomotion Using Low-Dimensional Planning. *ACM Transactions on Graphics*, 29(3), 2010.

[57] Igor Mordatch, Martin de Lasa, and Aaron Hertzmann. Robust physics-based locomotion using low-dimensional planning. *ACM Transactions on Graphics*, 29(4):71:1–71:8, July 2010.

[58] Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Trans. Graph.*, 31(4):43:1–43:8, July 2012.

[59] Eadweard Muybridge. *Animals in motion / Eadweard Muybridge ; edited by Lewis S. Brown*. London,, 1925.

[60] J.A. Okun and S. Zwerman. *The VES Handbook of Visual Effects: Industry Standard VFX Practices and Procedures*. Focal Press. Elsevier Science, 2010.

[61] Ana Paul and Josh C. Bongard. The road less travelled: Morphology in the optimization of biped robot locomotion. In *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2001*, pages 226–232. IEEE Press, 2001.

[62] Zoran Popović and Andrew Witkin. Physically based motion transformation. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 11–20, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[63] Marc H. Raibert and Jessica K. Hodgins. Animation of dynamic legged locomotion. *SIGGRAPH Comput. Graph.*, 25:349–358, July 1991.

[64] Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael F. Cohen. Efficient generation of motion transitions using spacetime constraints. pages 147–154, 1996.

[65] Alla Safonova, Jessica K. Hodgins, and Nancy S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Trans. Graph.*, 23(3):514–521, 2004.

[66] Hyun Joon Shin and Hyun Seok Oh. Fat graphs: constructing an interactive character with continuous controls. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 291–298, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[67] Takaaki Shiratori, Hyun Soo Park, Leonid Sigal, Yaser Sheikh, and Jessica K. Hodgins. Motion capture from body-mounted cameras. *ACM Transactions on Graphics*, 30(4), August 2011.

[68] Karl Sims. Evolving virtual creatures. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22, New York, NY, USA, 1994. ACM.

[69] Kwang Won Sok, Manmyung Kim, and Jehee Lee. Simulating biped behaviors from human motion data. *ACM Transactions on Graphics*, 26(3):107:1–107:9, 2007.

[70] Kwang Won Sok, Katsu Yamane, Jehee Lee, and Jessica Hodgins. Editing dynamic human motions via momentum and force. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 11–20, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[71] Manoj Srinivasan and Andy Ruina. Computer optimization of a minimal biped model discovers walking and running. *Nature*, 439(7072):72–75, Jan 2006.

[72] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts, 1998.

[73] Jie Tan, Yuting Gu, Greg Turk, and C. Karen Liu. Articulated swimming creatures. *ACM Trans. Graph.*, 30:58:1–58:12, August 2011.

[74] Emanuel Todorov. A convex, smooth and invertible contact model for trajectory optimization. In *ICRA*, pages 1071–1076. IEEE, 2011.

[75] Adrien Treuille, Yongjoon Lee, and Zoran Popović. Near-optimal character animation with continuous control. *ACM Trans. Graph.*, 26(3):7, 2007.

[76] Michiel van de Panne. Parameterized gait synthesis. *IEEE Comput. Graph. Appl.*, 16(2):40–49, 1996.

[77] Kevin Wampler, Erik Andersen, Evan Herbst, Yongjoon Lee, and Zoran Popović. Character animation in two-player adversarial games. *ACM Trans. Graph.*, 29(3):26:1–26:13, July 2010.

[78] Kevin Wampler, Daichi Sasaki, Li Zhang, and Zoran Popović. Dynamic, expressive speech animation from a single mesh. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '07, pages 53–62, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[79] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. Optimizing walking controllers. *ACM Trans. Graph.*, 28:168:1–168:8, December 2009.

[80] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. Optimizing walking controllers for uncertain inputs and environments. *ACM Trans. Graph.*, 29:73:1–73:8, July 2010.

[81] Jack M. Wang, Samuel R. Hamner, Scott L. Delp, and Vladlen Koltun. Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Trans. Graph.*, 31, 2012.

[82] Xiaolin Wei, Jianyuan Min, and Jinxiang Chai. Physically valid statistical models for human motion generation. *ACM Trans. Graph.*, 30:19:1–19:10, May 2011.

[83] Greg Welch and Eric Foxlin. Motion tracking: No silver bullet, but a respectable arsenal. *IEEE Comput. Graph. Appl.*, 22(6):24–38, November 2002.

[84] Andrew Witkin and Michael Kass. Spacetime constraints. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 159–168, New York, NY, USA, 1988. ACM.

[85] Andrew P. Witkin and Zoran Popović. Motion warping. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, pages 105–108, August 1995.

[86] H Woltring. New possibilities for human motion studies by real-time light spot position measurement. *Biotelemetry*, 1(3):132–146, 1974.

[87] W. L. Wooten and J. K. Hodgins. Simulating leaping, tumbling, landing and balancing humans. *International Conference on Robotics and Automation (ICRA)*, pages 656–662, 2000.

[88] Jia-chi Wu and Zoran Popović. Realistic modeling of bird flight animations. *ACM Trans. Graph.*, 22:888–895, July 2003.

[89] Jia-chi Wu and Zoran Popović. Terrain-adaptive bipedal locomotion control. *ACM Transactions on Graphics*, 29(4):72:1–72:10, Jul. 2010.

[90] KangKang Yin, Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Continuation methods for adapting simulated skills. *ACM Trans. Graph.*, 27(3):1–7, 2008.

[91] KangKang Yin, Kevin Loken, and Michiel van de Panne. Simbicon: Simple biped locomotion control. *ACM Trans. Graph.*, 26(3):Article 105, 2007.

[92] M Y Zarrugh and C W Radcliffe. Predicting metabolic cost of level walking. *Eur J Appl Physiol Occup Physiol*, pages 215–223, 1978.

[93] Li Zhang, Noah Snavely, Brian Curless, and Steven M. Seitz. Spacetime faces: High-resolution capture for modeling and animation. In *ACM Annual Conference on Computer Graphics*, pages 548–558, August 2004.

Appendix A

JOINT ROTATION IMPLEMENTATION

The animals appearing in this thesis are constructed from a kinematic tree of limbs connected by joints. The rotation at each joint is parameterized by the joint's *degrees of freedom*. In my implementation there are several different ways in which this parameterized rotation can be defined:

- fixed (0 DOFs)
- hinge (1 DOF)
- Euler angles (2 or 3 DOFs)
- exponential map (3 DOFs)
- free (6 DOFs)

There are a few subtleties to implementing some of these joints in a manner that allows easy differentiation of the resulting transformations, so I will cover each joint in turn.

Fixed: A fixed joint rigidly attaches two limbs. The transformation matrix resulting from the joint's rotation is thus simply a constant 3×3 user-specified matrix.

Hinge: A hinge joint allows rotation about a single user-specified axis, and is useful for modeling knee and elbow rotations. This rotation is parameterized by a single degree of freedom, denoted θ. If x, y, and z represent the components of the axis of rotation and

$x^2 + y^2 + z^2 = 1$, then the resulting rotation matrix is:

$$\begin{bmatrix} x^2(1-c)+c & xy(1-c)-zs & xz(1-c)+ys \\ xy(1-c)+zs & y^2(1-c)+c & yz(1-c)-xs \\ xz(1-c)-ys & yz(1-c)+xs & z^2(1-c)+c \end{bmatrix} \tag{A.1}$$

where $s = \sin(\theta)$ and $c = \cos(\theta)$.

Euler Angles: This type of joint can be of either two or three DOFs, and is simply represented by the composition of several hinge joints about user-specified axes.

Exponential Map: Joints parameterized by an exponential map describe an arbitrary rotation parameterized by three DOFs, denoted x, y, and z. Together x, y, and z describe a vector, with the length of this vector denoting the angle of rotation and the direction of this vector denoting the rotation axis. The most straightforward way of implementing this transformation contains a discontinuity at $x = 0$, $y = 0$, $z = 0$. Fortunately this discontinuity can be removed by reformulating the transformation.

Let $\theta = \sqrt{x^2 + y^2 + z^2}$, $s = \text{sinc}(\theta)$, and $c = \text{cosc}(\theta)$. Then the 3×3 matrix representing the joint's rotation can be written as:

$$\begin{bmatrix} x^2c + 1 - c\theta^2 & xyc - zs & xzc + ys \\ xyc + zs & y^2c + 1 - c\theta^2 & yzc - xs \\ xzc - ys & yzc + xs & z^2c + 1 - c\theta^2 \end{bmatrix} \tag{A.2}$$

Where $s = \text{sinc}(\theta)$, and $c = \text{cosc}(\theta)$. The sinc and cosc functions are defined as follows:

$$\text{sinc}(\theta) = \begin{cases} \frac{\sin(\theta)}{\theta} & \theta \neq 0 \\ 0 & \theta = 0 \end{cases} \tag{A.3}$$

and

$$\text{cosc}(\theta) = \begin{cases} \frac{1-\cos(\theta)}{\theta^2} & \theta \neq 0 \\ \frac{1}{2} & \theta = 0 \end{cases} \tag{A.4}$$

The sinc and cosc functions also have derivatives defined by:

$$\frac{\partial}{\partial\theta}\text{sinc}(\theta) = \begin{cases} \frac{\theta\cos(\theta)-\sin(\theta)}{\theta^2} & \theta \neq 0 \\ -\frac{1}{3}\theta & \theta = 0 \end{cases} \tag{A.5}$$

$$\frac{\partial}{\partial\theta}\text{cosc}(\theta) = \begin{cases} \frac{\theta\sin(\theta)+2\cos\theta-2}{\theta^3} & \theta \neq 0 \\ -\frac{1}{12}\theta & \theta = 0 \end{cases} \tag{A.6}$$

When writing an implementation for sinc, cosc, and their derivatives it is important to account for the error introduced when $\theta \approx 0$. To minimize this error, I use the following definitions in my implementation:

$$\text{sinc}(\theta) \approx \begin{cases} \frac{\sin(\theta)}{\theta} & |\theta| > 10^{-3} \\ 1 - \frac{\theta^2}{6} & |\theta| \leq 10^{-3} \end{cases} \tag{A.7}$$

$$\text{cosc}(\theta) \approx \begin{cases} \frac{1-\cos(\theta)}{\theta^2} & |\theta| > 5 \cdot 10^{-3} \\ \frac{1}{2} - \frac{\theta^2}{24} & |\theta| \leq 5 \cdot 10^{-3} \end{cases} \tag{A.8}$$

$$\frac{\partial}{\partial\theta}\text{sinc}(\theta) \approx \begin{cases} \frac{\theta\cos(\theta)-\sin(\theta)}{\theta^2} & |\theta| > 5 \cdot 10^{-4} \\ -\frac{1}{3}\theta & |\theta| \leq 5 \cdot 10^{-4} \end{cases} \tag{A.9}$$

$$\frac{\partial}{\partial\theta}\text{cosc}(\theta) \approx \begin{cases} \frac{\theta\sin(\theta)+2\cos\theta-2}{\theta^3} & |\theta| > 5 \cdot 10^{-3} \\ -\frac{1}{12}\theta & |\theta| \leq 5 \cdot 10^{-3} \end{cases} \tag{A.10}$$

Free A free joint is only used to describe the root motion of an animal and differs from the previously defined joint parameterizations in that it provides both a rotation and a translation. The joint's transformation is parameterized by six degrees of freedom. These are divided into three DOFs describing the rotational component through an exponential map and the remaining three DOFs describing the translation along the x-, y- and z-axes.

Appendix B

CEC 2006 RESULTS

This table summarizes the results of the basin-CMA optimization method when applied to the 2006 IEEE Congress on Evolutionary Computation (CEC-06) constrained real parameter optimization competition. The efficiency of my method in finding each problem's global minimum is measured by the ratio of the number of function evaluations of the top scoring CEC contestant over that required by basin-CMA. Thus a ratio of 10 means that basin-CMA method required ten times fewer function evaluations. Note that the best CEC contestant is chosen on a *per problem* basis, so no single algorithm in the competition would rate as well against basin-CMA and the table indicates. Since basin-CMA requires the use of derivatives, I provide two efficiency ratios. fevals ratio_1 gives a best case ratio where derivative solutions only count for a single additional function evaluation, while fevals ratio_2 gives a worst-case ratio when finite differences is used. The opt? column describes if basin-CMA correctly found the problem's global optimum. The rate at which CMA found the global optima here is on par with the best of the CEC-06 entrants. If basin-CMA is instead run with parameters such that the time required to perform the optimizations is on par with the best CEC-06 entrant, then its success rate at finding the global optima exceeds that of any entrant.

#	dimension	opt?	fevals ratio$_1$	fevals ratio$_2$
1	13	yes	105.970464	7.569319
2	20	yes	10.735468	0.511213
3	10	yes	20.546281	1.867844
4	5	yes	29.500000	4.916667
5	4	yes	59.019391	11.803878
6	2	yes	24.537736	8.179245
7	10	yes	89.488215	8.135292
8	2	yes	0.269050	0.089683
9	7	yes	6.083616	0.760452
10	8	yes	11.086012	1.231779
11	2	yes	12.345679	4.115226
12	3	yes	16.769231	4.192308
13	5	yes	29.487110	4.914518
14	10	yes	39.905063	3.627733
15	3	yes	40.223077	10.055769
16	5	yes	4.163090	0.693848
17	6	no	40.748068	5.821153
18	9	yes	109.538760	10.953876
19	15	yes	19.098863	1.193679
20	24	no	–	–
21	7	yes	26.338387	3.292298
22	22	no	–	–
23	9	yes	442.150171	44.215017
24	2	yes	2.457534	0.819178

Appendix C

SPACETIME CONSTRAINTS FOR CONTROLLER OPTIMIZATION

The spacetime constraints solver used to generate the individual motions used to create motion-graph based controllers (chapter 5) is based on the formulation described earlier in chapter 3. When generating cyclic motions, the constraint functions and the biomechanical component of the cost in equations 3.1 and 5.1 may be used directly from their description. For motions intended to connect states within an existing motion graph, it must be ensured that the generated motion transition seamlessly from its start and end states in the graph. To achieve this it is sufficient to ensure that the first and last pairs of frames each match a pair of consecutive states in the graph. Therefore the optimization variables corresponding to the first two and the last two frames are fixed and only the variables associated with the interior frames are optimized over.

It remains to combine the biomechanical and task costs into the objective function as in equation 5.1. To do this, first the initialization motion $\widetilde{M}$ is attached to the current graph $\mathbf{G}$. Then a set of samples is generated in the state space of the controller's MDP distributed according to the controller's influence function using the method described at the end of section 5.2.1, and the m samples are retained which lie at frames in $\widetilde{M}$. The objective function can then be calculated as:

$$\frac{1}{n}\sum_{f_i}\left[\alpha\ \log(cost_b(M(f_i))) + \frac{1}{m}\sum_{s_j=(f_i,r_j,\theta_j)} cost_t(s_j,\pi^*(s_j))\right] \tag{C.1}$$

This is equivalent to computing equation 5.2 just along the motion being optimized. Here $cost_b(M(f_i))$ is the objective function as described in chapter 3 and α is a weighting term setting the tradeoff between minimizing biomechanical versus task costs. I typically employ values of α from 0.1 to 1. The logarithmic scaling of the biomechanical cost is used to make it easier to define the task costs. For instance a gallop may require significantly

more energy than a slow walk, but that should not outright prevent the optimization from ever using such motions.

As a practical note, I have observed that the motion quality is not too much altered by simply using $cost_b(M(f_i))$ as the objective function for the spacetime constraints optimization and ignoring the task costs, so simply choosing this (less mathematically elegant) route is reasonable in many cases. This is because the task costs are incorporated at the motion clip level anyway, and the duration of clips is short enough that the within-clip deviation from the truly optimal motion is small. Since the optimization resulting is slightly more robust than that defined by equation C.1, I recommend this as a reasonable place to sacrifice mathematical elegance for practicality if one is having trouble getting the spacetime optimizations to converge. Nevertheless I have found the optimization from equation C.1 to be robust enough to use effectively in practice.

VITA

Kevin Wampler received his B.S in Computer Science and in Mathematics from the University of Arizona in 2003. After spending one year in the Masters of Computer Science at the University of Arizona, he moved to Seattle to join the graduate program in Computer Science and Engineering at the University of Washington. Toward the end of his first year in the graduate program at the University of Washington he began research with Zoran Popović. In 2012 he received his Doctor of Philosophy degree.