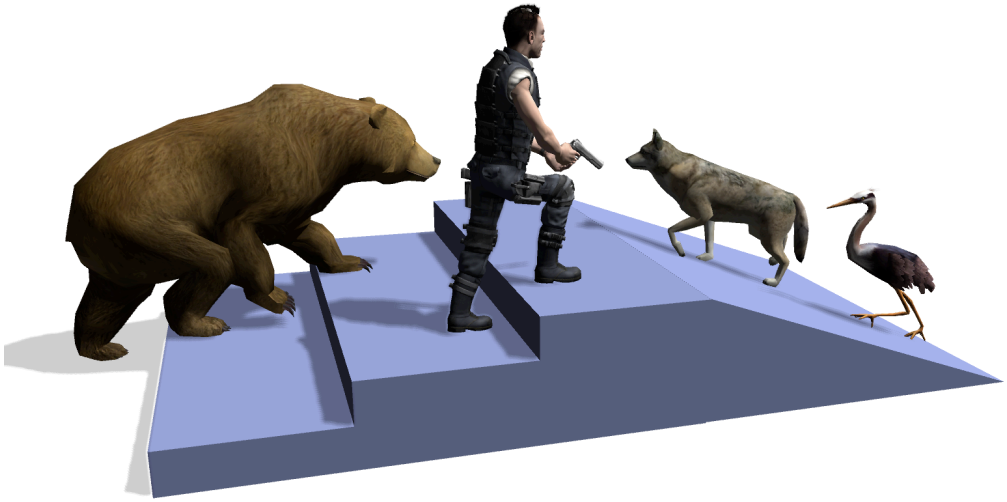
Master's Thesis
Department of Information and Media Studies
Aarhus University

Automated Semi-Procedural Animation for Character Locomotion



Rune Skovbo Johansen
(20020182)

May 25, 2009

Abstract

This thesis presents a framework of techniques for interactive synthesis of highly flexible character locomotion. The system uses a set of example motions primarily in the form of keyframed or motion-captured walk and run cycles. The system automatically analyzes each motion at design-time and extracts parameters such as impact and lift-off times for each foot as well as overall velocity. At runtime the system first blends the motions according to the current velocity and rotational velocity of the character, it then adjusts the movements of the bones in the legs by means of inverse kinematics to ensure that the feet step correctly on the ground. The system works for both human and non-human characters with any amount of legs and in whatever style the provided example motions are in. It can adjust animations made for a specific speed and direction on a plain surface to any speed, direction, and curvature, on any surface, including arbitrary steps and slopes.

Notable innovations in the thesis include the introduction of the concept of a footbase, which is a single combined heel and toe constraint that can retain the important information about the alignment of a foot relative to the ground; the calculation of a *supporting ground height* that can be used to produce motion with a good sense of weight, regardless of the number of legs and of the gait style; the calculation of natural-looking foot alignments that works for just about any characters and gait styles, and a scattered data interpolation algorithm that has desirable properties when interpolating motions with different velocities.

Acknowledgments

First of all thanks to the people at Unity Technologies, for believing in the potential of my ideas early on, lending me excellent hardware and software to develop on, and for providing the remarkable Unity game engine that gave my productivity a serious boost. A special thanks to Joachim Ante, for coming with answers and suggestions on in-depth technical matters about the game engine and for helping bridging contacts to other people.

Thanks to eduweb and WolfQuest.org for lending me high-quality animated quadrupled animal models to test with, specifically the wolf, coyote, and grizzly bear models used in this work.

Thanks to Tom Forsyth, whom I first met at GDC 2008, for showing me and discussing his demo How To Walk that has been one of the sources of inspiration for this thesis.

Thanks to Alex J. Champandard, Lucas Kovar, and Michael Gleicher for thoroughly discussing with me all the gritty details of sampling of parameter space in motion interpolation.

Thanks to Lars Kroll Kristensen for feedback and advise on presentational aspects of the thesis.

Thanks to the user community around Unity, for always being friendly, helpful, and resourceful.

Thanks, not least, to my supervisor Jesper Mosegaard for guidance and encouragement throughout the thesis.

To Marie-Louise, my friends, and my family: Thank you for all your patience, support, and understanding!

Electronic Material

This thesis presents methods for interactive synthesis of character animation and would not be complete without a demonstration of the capabilities of the implemented system in action. The encased CD-ROM included with this thesis contains videos as well as interactive demos demonstrating the system.

The videos can serve as an excellent introduction to this thesis. Certain parts of the thesis may be simpler to understand if the videos have been watched in advance, since things relating to movement is inherently troublesome to fully present with static text and images alone.

The implemented system can be tested, toyed with, and experienced first-hand in the interactive demos. The demos require the installation of a browser plug-in distributed by Unity Technologies ApS.

In the event that no CD-ROM is available, the electronic material can also be accessed online at:

```
http://runevision.com/thesis/
```

Contents

Chapter 1

Introduction

The market of computer games is abundant with games featuring highly detailed realistic or stylized characters perforing tasks in equally highly detailed and often very dynamic virtual worlds. However, the quality of the motions that these characters perform are rarely up to the same standard of quality and believability as the graphical appearance. Characters are often seen walking and running with feet that do not quite manage to stay fixed on the ground, and particularly when walking on steps and slopes, this may cause them to appear to be floating around, rather than being properly physically situated in the environment.

It is widely aknowledged that while graphical rendering and physical simulation has reached high standards in modern games, animation is generally lacking behind and is currently one of the main factors holding back an overall higher level of immersion. While some developers with enough ressources to back it have taken strides to develop games with a high quality of animation too, this task is generally ressource-hungry, and not widespread yet.

1.1 Motivation

Animation in games is most often created as generalized synthesized motion interpolated from a number of example motions, that have been created from motion-capture or animated by hand. These data-driven approaches have been remarkably successful at generating high-quality animated motion as long as the required flexibility of the synthesized motion is limited. However, state of the art animation today require very large amounts of example motions to interpolate from. This can be problematic for smaller developers with limited resources, but even when plenty of resources are available, the continued rise in demand for ever increasing flexibility of the synthesized motion necessitates example motions in quantities that are exponentially growing. Throwing more example motions at the problem is not a sustainable solution. Methods are needed that instead are good at generalizing and adjusting example motions beyond mere interpolation. Such methods can be designed to synthesize specific types of motion in a very flexible manner, based on a deeper understanding of those types of motions.

Since character locomotion such as walking and running is one of the types of motions used in most games and which players are typically most exposed to throughout a game, character locomotion has been chosen as the focus of this thesis. Many existing methods have been proposed for generating flexible character locomotion without the need of excessive amounts of motion. Procedural methods can create motion sometimes without needing any example motions at all, but such fully procedural approaches lack efficient control over style and personality. Motion capture is widely used for creating highly realistic example motions, while animators are trained in, and very good at using animation software to create motions of high quality with a given desired personality and style. Any approach not utilizing these technologies and skills optimally is not suitable for the majority of animation in modern games of today. Dynamic simulations of physics is gaining popularity in games, but do not by themselves provide enough extra flexibility of motion. For example, an important part of believable locomotion is anticipation of where to step next, and physical simulation does not help solve that problem. Learning based methods can be used to adapt physically simulated example motions to new situations, but this currently is a complicated process that require a non-trivial investment from the user. Also, simulation based approaches generally exclude styles of motion that are not supposed to be realistic.

An approach rooted in synthesized motion based on example motion and augmented with procedural methods has been found ideal for producing the needed flexibility while taking fully advantage of example motions created by animators or from motion-capture. The goal of this thesis is to create a framework of techniques for automated semi-procedural animation that can be used to obtain flexible and believable character locomotion, while the amount of needed example motions and the time and expertise spent on setting up the animation framework can be kept to a minimum.

1.2 Problem Statement

Computer games with ever more realistic graphics and dynamic environments need animated characters that can adapt to and respond to these dynamic environments in a believable fashion. Among the various needs, believable locomotion (walking and running) is the most common, but it is difficult to get right. Furthermore, it needs to be as simple as possible to achieve for the animator and programmer.

In order to ease the wide accept of a new technology, it needs to support existing work-flows. While experiments with purely procedural animation are occasionally conducted, the de facto standard in the industry is the use of key-framed animations (example motions) made from motion-capture or animated by hand, since this allows for the greatest artistic control. For continuously repeating motions such as walking or running, cyclic animations are typically used. Furthermore a new technology is easier embraced if it requires a minimum of extra work and effort to use.

Many game developers, particularly small to medium sized, would benefit from

a framework of techniques for real-time animated character locomotion based on key-framed animation cycles made from motion-capture or animated by hand, that automatically adjusts the motion of a character to the terrain at runtime. In other words a framework of techniques is needed for easily setting up semi-procedural animation for character locomotion.

Problem Creating flexible and believable animation of arbitrary characters walking and running in games involves a large amount of work creating the example motions and setting up a suitable animation framework for the game.

Hypothesis By using a framework of techniques for automated semi-procedural animation, it is possible to obtain flexible and believable animation while the amount of needed example motions and the time and expertise spent on setting up the animation framework can be kept to a minimum.

Method Implementation of a framework of techniques that automatically identifies key stylistic and functional aspects of provided example motions and utilizes these at runtime for synthesis of semi-procedural animation. The believability of the synthesized motion will be evaluated, as well as the genericness for use with arbitrary characters, the flexibility of the synthesized motion, and the ease of use of the framework.

Limitations The framework of techniques is based strictly on cyclic walk and run cycles as example motions, where the character is walking on the spot with a would-be moving ground beneath. In the implemented framework only example motions on a flat horizontal plane are considered, though the synthesized motion can be on any terrain.

Note that *believable* motion is chosen as a criteria rather than realistic or lifelike. In a game with realistic graphics and animation the synthesized motion should also be realistic to be believable. However, in a more stylized game with for example exaggerated movements, the synthesized motion should fit that style as well. Thus, believable motion may be a measure of realism in some cases but it should be more generally understood as roughly equivalent to animation craftsmanship[1].

The given problem is characterized by requiring many orthogonal goals to be met that are relevant to different people and must be evaluated in different ways and none of which can be ignored. To provide a clear overview of the multiple challenges that must all be addressed, a set of framework goals has been stated to provide better clarity of the multi-faceted problem.

1.2.1 Stated Framework Goals

Since not all games revolve around only human or humanoid characters, or take place in a world with realistic physics, the system should be sufficiently *generic* to work with characters with almost any morphology and style:

[1] It must be assumed that the highest goal that synthesized motion in games can aspire to is to look as if it had been carefully crafted by a skilled animator in whatever style the game has.

- It should work for practically any biped or non-biped characters.
- It should work with any style inherit in the example animation cycles provided.

Since few assumptions can be made about in which directions and on which terrain the characters in a game move around, the system should be able to synthesize *flexible motion* with the following properties:

- It should support straight, curved, sidestepping, and backwards motion, and anything in between.
- It should adapt dynamically to uneven terrain including arbitrary steps and slopes.
- It should transition naturally between standing, walking, and running.

The system should be cheap and *easy to use* with regard to the artist and programmer resources needed to use it:

- It should need just a few example animation cycles, not an extensive motion library.
- It should be very quick to setup initially and need only a minimum of parameter tweaking.
- It should be controlled at a high level by simply specifying the position and facing direction of the character (which amounts over time to the velocity and rotational velocity).

The system should be *faithful*:

- It should produce motion for the characters that are exactly like the example motions when possible.
- It should move the character around in exact accordance with the specified position and facing direction.

The system should produce *believable* motion:

- The synthesized motion should be *believable* in a way that naturally extends the provided example motions.

The system should be *efficient*:

- The system should use as few CPU resources as possible so that it can scale well to multiple characters.

1.3 Key Terms and Concepts

The following present an overview of selected key terms and concepts that are central to understanding this thesis and related work in the area.

1.3.1 Character Locomotion

Locomotion is the self-propelled movement of an organism, such as walking or running of a character with legs.

A character is defined in this work as an animated legged human or non-human 3D character in the context of a 3D computer game or virtual world. Typically characters are controlled either by a player as an avatar, or by the computer as an NPC (non player character) with artificial intelligence (AI). In some games a character can be controlled to walk or run in various directions by pressing the arrow keys on the keyboard, or by a game-pad, joystick, or similar controller. In other games the player can click with the mouse on the ground after which the character will walk towards that point, sometimes autonomously finding its way around obstacles. Either way, some logic is used to translate the player or AI control input into simple movement commands used to make the character move.

Typically, a character is represented abstractly in the game as a single rigid body *object* with a location and orientation in the world. Visually, the character may have any shape, and this shape may be meticulously animated, but to most of the game logic the character is still just a single object. The previously mentioned movement commands will typically move the character object around by applying a force or velocity to it in a certain direction, causing its position to move over consecutive frames. Only rarely do games or virtual worlds bother with an actual physical simulation of leg muscles to drive the movements of the legs and drive the character forward.

In this work a character is assumed to be represented as an simple object with a position and orientation in the world as well as a potentially complex animated visual representation. No particular method of moving the character object around is assumed; only that it will result in the position and orientation of the character object to move over consecutive frames. The character locomotion that this work is concerned with is to make the animation of the visual representation of the character look good when the character is walking or running around.

1.3.2 Skeletal Animation

Skeletal animation is a technique in computer animation, particularly in the animation of humans and creatures (here referred to as *characters*). In skeletal animation, a character is represented in two parts: A surface representation used to draw the character (called the skin) and a hierarchical set of *bones* (rotational joints) used for animation only (called the skeleton).

Except for the root bone, each bone in the skeletal hierarchy has a parent bone

and a three dimensional transformation that is relative to the transform of the parent. The full transform of a child node is the product of its parent transform and its own transform, so for example moving a thigh-bone will move the lower leg and foot too. The root bone has a transform in global space.

Characters in today's games typically have a number of animations of limited duration stored, often referred to in academia as *motions*. One motion may for example have the character standing still (being idle), while another may have the character walking. Motions are created by animators and may be based on motion-capture. Each motion stores the movements over time of each bone in the skeleton.

1.3.3 Walk and Run Cycles

Some motions, typically motions for walking and running, can be constructed in such a way that they are perfectly cyclic. In a cyclic motion the configuration of the bones at the end of the motion matches the configuration at the beginning, such that when the motion is played repeatedly, the result is a smooth continuous motion. Cyclic motions for walking and running are referred to as walk cycles and run cycles respectively. Note that for the purposes of this work, no distinction is made between walk and run cycles. The difference is here considered purely stylistic between walking and running, as well as jogging, sneaking, and other cyclic motions that involve the character taking a step with each foot. Walk and run cycles are widely used i games since they make walking or running of arbitrary duration possible based on just a few short stored motions.

1.3.4 Animation Blending

In today's games and other real-time applications with animated characters, animation blending is often used for several purposes. Blending enables the bones of a character to not just be moved by a single stored motion, but rather be moved simultaneously by multiple motions. At any given time each motion is given a weight which is used in a weighted average to create the combined, blended motion. These weights may be continuously changed over time.

Blending can be used to ensure that characters move smoothly when transitioning from one motion to another. Rather than abruptly stopping playing the current motion and starting playing the next, the weight of the current motion can be gradually decreased from one to zero while the influence of the next motion is simultaneously increased from zero to one. This technique is referred to as cross-fading.

Another use for animation blending is the synthesis of motions with properties different from any of the stored motions. For example, a character may have a stored walk cycle for walking straight ahead and another stored walk cycle for turning sharply to the right with a certain turning angle. By blending these two motions, each with a given weight, a new walk cycle can be synthesized that can have a less sharp turning angle. In academia, the motions that are blended together are often referred to as *example motions*.

1.3.5 Keytimes

Certain methods of motion synthesis may need to know certain properties of the provided example motions. Keytimes signify the times of certain important events in the motion. For example, walk and run cycles may be annotated with keytimes such as *toe-off* and *heel-strike* to mark the times when a foot lifts off the ground and strikes the ground again, respectively. Keytimes for a motion may be annotated manually by animators. In some cases it is possible to identify keytimes automatically using heuristics.

1.3.6 Footprint / Footplant / Foothold

Footskate is an artifact sometimes seen in character animation; particularly in computer games. It refers to the phenomenon that the feet of an animated character may seem to not stay properly fixed on the ground, but rather slide along the ground at times when they are not supposed to. Certain methods of motion synthesis for character locomotion explicitly prevent footskate by ensuring that the feet of a character are standing firmly on the ground when grounded. This is often done by marking a position and orientation on the ground, called a *footprint*, *footplant*, or *foothold*, where a foot is supposed to stay fixed for a duration of time, and then enforce that constraint. Since footplants are often the primary connection between a character and the surrounding environment, even small amounts of footskate can destroy the realism of a motion.

1.4 Delimitation

This thesis presents a model for for legged locomotion that can be applied to a very wide range of legged characters and styles while also producing motion that is very flexible. However, certain assumptions are made about the skeletons of the character and about the example motions provided.

The framework of techniques is based strictly on cyclic walk and run cycles as example motions, where the character is walking on the spot with a would-be moving ground beneath. In the implemented framework only example motions on a flat horizontal plane are considered, though the synthesized motion can be on any terrain. Over the duration of a motion cycle each leg should take exactly one step. This allows for almost all styles of gait, but excludes a few, such as skipping. The feet of a character should be at their lowest when resting on the ground, and when in contact with the ground, feet should be moving with an approximately constant velocity relative to the character. Furthermore, all the feet should be moving with the same velocity. It is assumed that only the rotation of each bone is animated while the position and scale is not[2]. This allows for all types of rotational joints but excludes for example sliding joints (such as hydraulic pistons) and stretchable

[2]However, note that due to the hierarchical nature of the bones, the rotation of a bone causes the positions in global space of all its descendant bones in the hierarchy to change.

limbs. However, these assumptions have been made mostly to simplify and optimize the implemented system, and are not an inherent restriction in the overall methods presented.

The system does not support a variable number of legs for one character. For example, it cannot be used for a creature that sometimes walk and two legs and other times crawl on four legs. This is more of a limitation in the implementation than in the general methods, although certain calculations would need to be slightly generalized in order to support a variable number of legs.

While inverse kinematics have an important function in the presented methods, the exact details of the algorithm are not the focus of this work. Only a simple inverse kinematics solver has been used, which will not be described in details.

1.5 Overview of the Thesis

In chapter 2 I describe the background and scientific environment of this thesis, and provide an overview of the general approaches dominating the field. The chapter also presents a summary of more specific techniques related to methods used in this thesis.

In chapter 3 I discuss the general approaches that dominate the field and I position this thesis in the related work.

The chapters 4 to 7 present the framework of techniques for automated semi-procedural animation for character locomotion. Each chapter describe a certain aspect of the framework in details while relating it to work done by others in greater details than in the chapter on Related Work.

Chapter 4 presents methods for automated analysis of motion cycle properties, such as keytimes, velocity, and foot trajectories.

Chapter 5 presents methods for blending multiple example motions in a synchronized fashion and discusses the approach taken in this work to providing high-level control of stylistic motion parameters.

Chapter 6 discusses various methods for scattered data interpolation as used for interpolation of example motions, and presents the method used in this work.

Chapter 7 presents methods for semi-procedural adjustments of walk and run cycles used to adapt a blended motion to an arbitrary velocity and rotational velocity on any uneven terrain.

Chapter 8 contains a summary of the main results presented in this thesis.

Finally, chapter 9 presents an outlook and ideas for future work, both in terms of research and of applications.

Part I

Problem Domain

Chapter 2

Related Work

Creating motions for characters in computer games poses many challenges. We are all accustomed to how characters move from our everyday: We have a life's worth of experiences with looking at how humans, and animals to a lesser degree, behave, move around, and perform various actions.

Living beings can do a large variety of different actions, and each action can be done in many ways often depending on context, such as the immediate environment. Believable characters in games must have this awareness and adaptation to context and environment to seem like they belong in the world they inhibit. At the same time we have all developed the ability from childhood to notice the subtleties of motion that convey personality and emotion. To model believable characters, these subtleties of motion must be expressed by the game characters.

On top of this comes the challenge of interactivity. Interactivity in games and simulations means that the future actions of characters cannot always be determined in advance, and character need to dynamically respond to the unfolding events in the game, whether controlled by the player, artificial intelligence, or something else. These dynamic responses often need to happen exceedingly quickly, which makes planning of motion ahead of time extra challenging.

2.1 General Approaches

A vast amount of research has explored animation techniques for character locomotion. Initial research has focused on offline motion editing, which seeks to help animators create flexible and new motion from existing motion examples. While this work is not directly applicable to realtime simulations such as games, some of the general concepts and techniques from these methods have found their way into realtime use in modified forms.

For realtime motion synthesis, there are a few different approaches to generating flexible and desirable character locomotion: Procedural motion (also called algorithmic or model-based motion), example-based motion, and dynamic simulation-based motion.

Animated characters are important in many types of computer games and interactive simulations. The movements of these characters are typically created by an example-based approach where pre-recorded clips of movement, either from motion capture or keyframing, are used in various combinations. Procedural approaches focuses on creating specialized techniques based on an understanding of movement. Example-based approaches use example motions as input and synthesize new motion from these, but may also additionally employ procedural techniques. Dynamic Simulation-based methods perform accurate physic simulations of character motion in order to increase both the situatedness and the realism of the motion. The simulation may be driven by example motions, procedural controllers, or a combination.

2.1.1 Offline Motion Editing

A lot of the research in motion capture has explored techniques for modifying data for a given scenario and for generalizing it for effective reuse. These techniques are not directly applicable to runtime motion synthesis but rather are methods for motion editing. Straightforward interpolation with keyframes, time-warping and motion displacement mapping was suggested initially (Bruderlin and Williams, 1995; Witkin and Popovic, 1995). Gleicher has improved edits and adapted motion to new characters by maintaining desired constraints such as contact with the environment while optimizing over an entire sequence (Gleicher, 1998). Addressing the same problem, Lee and Shin (1999) have presented an approach using multi-level B-splines for adapting existing motion of a human-like character, decoupling the problem into manageable sub-problems. Kovar et al. (2002) presented a method for footskate cleanup for motion capture editing based on identifying and enforcing foot constraints.

Path planning is a viable approach for motion editing, where future actions are known well in advance. An optimization-based scheme has been proposed by van de Panne (1997) to generate biped locomotion from given footprints. This scheme was further extended to quadruped locomotion (Torkos and van de Panne, 1998).Choi et al. (2003) have presented a scheme for planning natural-looking locomotion of a biped figure to facilitate rapid motion prototyping and task-level motion generation. Based on a combination of probabilistic path planning and hierarchical displacement mapping, the scheme gives a sequence of motions to move from a start to a goal position using a set of live-captured motion clips. Yamane et al. (2004) have explored an approach for animating characters manipulating objects that combines path planning with data-driven, constraint-based inverse kinematics. A path planner is used to find motions that satisfy geometric, kinematic, and posture constraints, while redundancy in poses are resolved by biasing the solution toward natural-looking poses extracted from a database of captured motions.

2.1.2 Purely Procedural Locomotion

Procedural methods focuses on creating specialized techniques based on an understanding of movement encoded into a model of how that movement can be simulated. The procedures can be as sophisticated as needed to provide different motion styles or react to different conditions of the simulated environment. The degree to which this method succeeds relies on how accurately we can understand and model human motions, or motions of whatever creatures that need to be modeled.

Bruderlin and Calvert (1993, 1996) have generated locomotion in realtime by simulating an inverted pendulum for a stance leg. An interactive control hierarchy was adopted to produce a variety of personalized human walking. kai Chung and Hahn (1999) has developed a procedural hierarchical motion control system to generate human locomotion along a path on uneven terrain.

The major problem with purely procedural motion is that it cannot take advantage of the skills that trained animators have to create expressive motion, or the power of using motion capture to retrieved highly realistic motion. This means that the nuances of traits such as personality and mood can be difficult to create. Focus in modern approaches seem to have shifted mainly to example based approaches, while dynamic approaches (using physics simulation) can be seen as a modern descendant to the purely procedural approaches.

2.1.3 Example-Based Locomotion

Example based motion synthesis uses sets of example motions with an interpolation scheme to construct new motions. Example based methods attempt to provide a set of meaningful, high-level control parameters to the animator or runtime system, while maintaining the aesthetic of the example motions in the blended motion. On top of this, some example based methods employ procedural techniques to increase the flexibility of the synthesized motion beyond the parameter space formed by the example motions alone.

Unuma et al. (1995) has applied Fourier analysis to motion data for interpolating and extrapolating the human locomotion such as walking motion. Wiley and Hahn (1997) have provided an interpolation technique for example motions by resampling the data in a grid in the parameter space. Among other uses, they used linear interpolation across a single parameter, slope angle, to generate walking on a sloped surface. Rose et al. (1998) has presented scattered data interpolation techniques which are suitable for blending example motions located irregularly in parameter space. Radial basis function interpolation and time-warping was used to generate motion with different emotional adverbs, such as happy and sad walking, while per-frame inverse kinematics was used to enforce labeled constraints.

Sun and Metaxas (2001) have used a hybrid approach, incorporating the use of multiple example motions coupled with procedural methods based on bio-mechanical knowledge to generate uneven terrain locomotion with adjustable gait parameters. Using their methods, a system can operate automatically, by knowing how step length or toe-out will affect turning radius, or how step height and step length in-

teract. Park et al. (2002) has presented a method for realtime character locomotion using motion blending based on scattered data interpolation. Using manually annotated keytimes, time-warping, and inverse kinematics, the motion is adapted to uneven terrain while also ensuring that there is no footskate. Meredith and Maddock (2005) have used automatically identified keytimes and inverse kinematics to retarget an input motion to different body proportions as well as uneven terrain.

All these methods for realtime locomotion are all based on parameterization of a few parameters, such as speed and turning angle, while taking steps in arbitrary direction like sideways or backwards is not supported. All the methods also require example motions for all controllable parameters, except the method by Park et al. (2002), which can retarget motion to uneven terrain without the need for example motions at different slope angles.

Forsyth (2004) has proposed a method for character locomotion that predicts future footprint positions and uses per-frame inverse kinematics to to adjust the motion to those footprint constraints.

In contrast, the methods in this thesis can generate locomotion with steps in arbitrary directions and on any arbitrary terrain including both steps and slopes, without the need of more than two example motions. However, when multiple example motions with different velocity parameters are present, the method does makes use of them to further improve the synthesized motion.

2.1.4 Dynamic Simulation-Based Locomotion

A number of papers present physics-based solutions for creating realistic locomotion on varied terrain. These approaches usually represent the bodies of characters as joint connected physical objects that can be manipulated by applying torque forces on the individual joints. Walk controllers are then created by continuously attempting to drive the joints towards their respective target angles using proportional-derivative (PD) controllers. This is in contrast to kinematic approaches, that manipulate positions, angles, and velocities without regard to which forces created them. Dynamics simulation approaches focus on generating physically feasible locomotion, and as such, usually cannot simulate unrealistic motions, for example where the center of mass is in a position such that the character cannot maintain its balance.

Komura et al. (2004) have proposed a technique for generating human reactive motion responses to external dynamic perturbations and maintain balance while walking or running. Wrotek et al. (2006) have presented an approach to controlling physically simulated animated characters in a dynamic virtual world based on example motions. Unrealistic but effective world-space torques are used for increased stability and a weak root spring for plausible balance. Though only in 2D, Sok et al. (2007) have created a technique for allowing physically-simulated biped characters to imitate human behaviors by transforming any example motion into a physically-feasible, balance-maintaining simulated motion. (Yin et al., 2007) have presented a walk controller that can produce robust motion in response to unanticipated downward steps of $20cm$ and slopes of ± 6 degrees.

Due to the lack of actual awareness of the local environment, larger steps and slopes are not well handled with any of the above simulation methods, and any upward step that is higher than the foot lifts up in the walk cycle will likely result in failure to proceed. Using continuation based optimization methods, Yin et al. (2008) have overcome this problem by creating techniques for generalizing a controller for physics-based walking to be able to perform different tasks, such as climbing a large $65cm$ step up, or pushing a heavy object. However, designing the search functions and choosing the appropriate set of optimization parameters requires a user in the loop and some experimentation (Yin et al., 2008, p5) for each of the new tasks that the character must be able to perform.

Besides the inability to simulate physically infeasible motions, methods based on physics simulation are typically computationally more expensive than approaches based on kinematics.

2.2 Specific Techniques

Besides the related work in the area of character locomotion in general, this thesis describes many specific techniques and methods for which there is also a large body of related work. Since the subject of that related work is very specific in nature and addresses problems that have not yet been properly introduced at this point in the thesis, the related work will be described in greater detail throughout the remaining thesis in dedicated sections in the relevant chapters. A very short summary of all the related work from the remaining thesis follows here.

Chapter 4 discusses motion analysis and the related work in that area. In order for motion blending based methods to handle provided example motions, it may need to know certain keytimes of those motions. In many works manually annotated keytimes are assumed (Rose et al., 1998; Lee and Shin, 1999; Park et al., 2002; Choi et al., 2003). Kovar and Gleicher (2004) has presented automated tools for locating logically similar motion segments, and Kwon and Shin (2005) has analyzed center-of-mass trajectories segment unlabeled motion sequences. Meredith and Maddock (2005) has used a gradient analysis technique for retrieving the flight path of a foot.

Chapter 5 discusses motion blending and the related work i that area. With a large number of example motions with different parameters, the task of assigning proper weights to a subset of the example motions can become quite complex. Unuma et al. (1995); Rose et al. (1998); Park et al. (2002); Kovar and Gleicher (2003) have used various techniques for interpolation and extrapolation of motions with different parameters and with time-warping. Gleicher (2008) discusses the problem of managing interpolation between sample motions in a multidimensional parameter space as the demand for versatile motion increases.

Chapter 6 discusses motion interpolation and the related work in that area. Motion interpolation is used to interpolate between multiple example motions, given a weight for each. The weights for the example motions are calculated using scattered data interpolation. A survey of often used scattered data interpolation methods has been presented by Amidror (2002). The technique of building parameterized mo-

tions from blends of captured examples was pioneered by Wiley and Hahn (1997) and Rose et al. (1998) and improved on by Sloan et al. (2001). The method has been critiqued by Allen et al. (2002, p617) and Kovar and Gleicher (2004, p561) who present their own respective improved techniques.

Chapter 7 discusses procedural techniques to augment example-based motion synthesis for character locomotion, as well as the related work in that area. kai Chung and Hahn (1999); Sun and Metaxas (2001) have developed different procedural motion control systems to generate walking motion along a path on uneven terrain. Gleicher (2001b) has discussed the implications of editing the path of a character while preserving the original constraints. Forsyth (2004) proposes a method that works for online interactive character motion. van de Panne (1997); Torkos and van de Panne (1998); kai Chung and Hahn (1999) have proposed various optimization-based schemes to generate locomotion from given footprints. Gleicher (2001a) has compared different constraint-based offline motion editing techniques. Gleicher (1998, p5) and Lee and Shin (1999) also advocate the use of smoothed displacement curves. kai Chung and Hahn (1999); Kovar et al. (2002) propose methods for natural-looking adjustments of leg poses, and Fêdor (2003) have compared different inverse kinematics algorithms.

Chapter 3

Positioning of This Thesis

As described in the previous chapter, purely procedural motion, example-based motion, and dynamic simulation-based motion present different general approaches to the problem of generating motion for character locomotion for use at runtime, such as in computer games. Of these, this thesis is rooted in the example-based approach augmented with procedural elements.

The major problem with fully procedural approaches is the lack of efficient control over style and personality, as well as the fact that different types of characters (such as animals with different amounts of legs) need different models. Since a method is desired here that is applicable to a wide range of different games, a fully procedural approach is deemed unfavorable.

The approach to motion that is based on dynamic simulations of physics is gaining popularity in games, though it is still used in only a minority of games. Simulation-based methods are computationally more expensive and by themselves do not provide much extra flexibility of motion. Learning based methods can be used to adapt example motions to new situations, but this currently is a complicated process that require a non-trivial investment from the user. Also, simulation based approaches generally exclude styles of motion that are not supposed to be realistic.

Traditional example-based approaches have have been remarkably successful at generating high-quality animated motion that is both true to the provided example motions and controllable with multiple parameters. However, motion that is controllable with only a discrete set of parameters fail to provide the flexibility needed for the large set of possible motions that might arise in a highly dynamic environment. This thesis seeks to pursue an approach to generating motion for character locomotion that overcomes some of the limitations with traditional example-based approaches while keeping the advantages by augmenting the example-based motion with procedural methods.

3.1 Necessity of Taking Arbitrary Steps

In the traditional example-based approach, multiple example motions can be blended to obtain an interpolated motion anywhere in between. If two example motions are

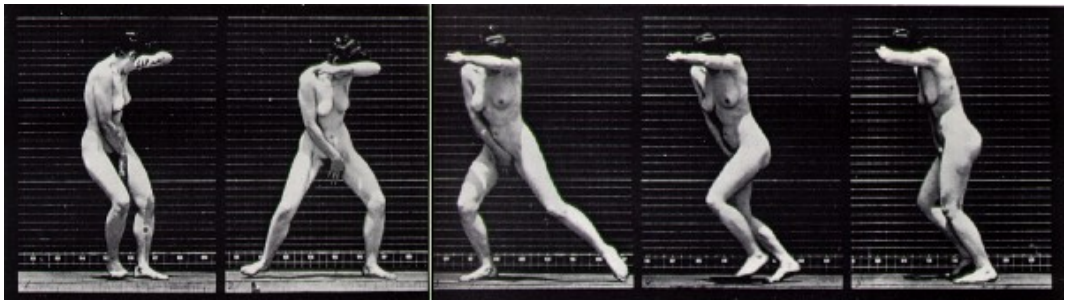


Figure 3.1: *"Turning around in surprise and running away" (1884-85) by Eadweard Muybridge, who is known for his early use of multiple cameras to capture motion. The motion depicts the woman taking steps in a specific direction long before she has fully turned around to face that direction.*

given with a character walking straight ahead, and turning sharply to the right, respectively, then blending can be used to make the character walk with any turning angle in between straight ahead and sharply to the right. Turning angle is then a *parameter*. There can be multiple motion parameters. Typical parameters for locomotion are speed and turning angle. Sometimes additional parameters are used for slope angle or for controlling emotional aspects of the motion.

However, in reality humans (and many animals) do not restrain themselves to walking with a specific speed and turning angle. Humans are not vehicles, but can take steps in arbitrary directions. Consider the scenario of a startled woman turning around and fleeing away from something (see the photos by Muybridge in figure 3.1). Such a person will not walk forward with a sharp turning angle (like a vehicle), nor will she neatly turn around on the spot before beginning to move. Rather, she will begin taking the first step backwards (in the direction she wishes to flee) and generally begin her movement in the desired direction long before she have fully turned around to face that direction. This behavior is not specific to startled woman, but generally applies to humans that need to suddenly go in a different direction that they happen to be facing. While quadrupled animals are slightly less agile in their ability to walk in directions independent from their facing direction, they too show the same behavior to some degree. This is just one illustration of the fact that locomotion in games need to go beyond speed and turning angle to obtain realistic movement, and allow steps at any time in arbitrary directions.

3.2 Insufficiency of the Discrete Parameter Space

Allowing characters to take steps at any time in arbitrary directions and on any uneven terrain require a flexibility of the motion that goes far beyond parameters for just speed and turning angle. The traditional approach to obtain better flexibility has been to add additional parameters to the motion. Generally, the number of needed example motions grows exponentially with the number of parameters, so this has often required very large number of example motions even for a relatively

low number of controllable parameters. Even so, this approach has still not been able to produce motion with quite the needed flexibility to allow arbitrary steps on arbitrary terrain.

It has been noted on several occasions that "[example-based] approaches fail to scale to the very large set of possible motions that might arise in a realistic environment. For example, there are an infinite number of ways in which two characters might bump into each other or in which a character may move through a constrained, unpredictable environment" (Yin et al., 2007, p1) and that "data-driven approaches are data hungry, particularly if the animation synthesis techniques perform only a shallow analysis of the motion capture data" (Yin et al., 2008, p1). Recently, the conclusion reached by many seems to be that "scaling up the number of examples is unlikely to sufficiently scale up the quality of the character animation. Methods that make better use of examples will be required" (Gleicher, 2008, p82).

An often mentioned solution is to combine example-based methods with procedural algorithms that make better use of the examples by employing domain-specific knowledge of the motions in question, since "algorithmic approaches have the potential to be more general and capable of generating families of motions rather than individual motions" (Yin et al., 2007, p1).

3.3 Minimal Model of Legged Locomotion

To overcome the limitations of example-based approaches, this thesis present methods to augment example-based interpolated motion with procedural techniques to make better use of the examples and generate more flexible motion. The challenge in this is to find the optimal balance between a model that is too general and one that is too specific. A model that is too general may fail to generate motion of appealing quality, or may leave many problems unsolved. For example, regular motion blending can be used for just about any motion and character, but will often produce artifacts such a footskate, and it still leaves the problem open of blending the right motions with the right timing. A model that is too specific may depend on knowledge that is only true for some characters. For example, the majority of research on procedural locomotion assumes a human character and may produce excellent and realistic motion for human characters, but cannot be used for any characters that are not human, or that are animated in a non-realistic style.

This thesis presents a model of motion for legged locomotion that can be applied to a very wide range of legged characters and styles while also producing motion that is very flexible. The assumptions that the model is based on are described in section 1.4 in the introductory chapter.

Part II

Locomotion System

Chapter 4

Motion Analysis

In this chapter methods are presented for automated analysis of properties of walk and run cycles, such as keytimes for each foot, the overall velocity of the motion, and foot trajectories. The analysis is primarily based on observations of the trajectories of the heel and toe, and information that can be derived from this. The analysis is performed at design-time, and the analyzed properties of the motions are stored such that they are readily available for the blending and semi-procedural methods at runtime.

4.1 Related Work

In order for procedural animation methods to handle provided example motions, it may need to know certain properties of those motions. Keytimes signify the times of certain important events in the motion, for example at which times in the cycle each foot lifts off the ground. The simplest way to obtain keyframes for a motion, though also the most time consuming, is to annotate the keytimes manually or interactively. This approach is taken in many works (Rose et al., 1998; Lee and Shin, 1999; Park et al., 2002; Choi et al., 2003).

(Kovar and Gleicher, 2004) has presented automated tools for locating logically similar motion segments in a data set and using them to construct parameterized motions. The tools do not employ explicit keyframes with a specific semantic meaning, but automatically find corresponding frames in the different motions by measuring the numerical similarity to a threshold value. Kwon and Shin (2005) has analyzed the center-of-mass trajectory of human walking and running motions to segment unlabeled motion sequences into motion half-cycles.

Meredith and Maddock (2005) has used a gradient analysis technique for retrieving the flight path of a foot from motion capture data. Over the course of the original motion, they analyze the height value of one of the character's feet. They interpret a switch from a negative to a positive gradient on a frame that is within 10% of the global minimum height for the foot to be the *foot-off* keytime. A switch back to negative gradient with a height value within 10% of the global maximum is interpreted as the peak of the flight height. Finally, a switch back to positive, again

within the 10% of the global minimum height, is interpreted as the *foot-strike*. In the data used for this work, analysis based on gradient changes alone proved to be fragile. Since the foot trajectory is mostly flat in the phase where the foot is resting on the ground, changes between negative and positive gradient can happen almost randomly based on small variations, imprecisions, or noise in the curves. This work uses instead an analysis of both the heel and toe, and tracks curvature changes in addition to gradient changes (see section 4.3.2 and 4.3.4).

Much previous literature (Rose et al., 1998; Lee and Shin, 1999; Park et al., 2002; Choi et al., 2003; Semančík et al., 2004) makes use of keytimes called *toe-off* and *heel-strike* to mark the times when a foot lifts off the ground and strikes the ground again, respectively. These can be misleading when used as general terms applied to any walk or run cycle, since even when only considering humanoid characters the heel may not always strike the ground first in certain styles of running. This was recognized by Bruderlin and Calvert (1996) but the same terms were used nonetheless. This work instead uses a different set of more general keytimes, as described in section 4.2.1.

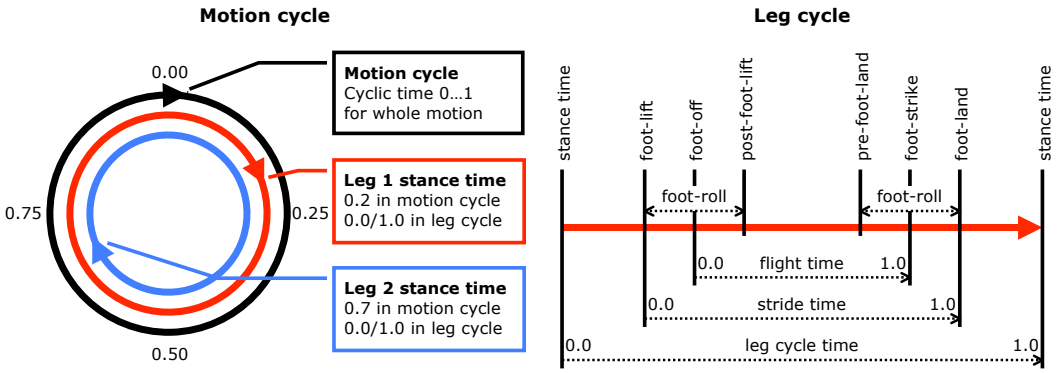
4.2 Motion Cycles and Leg Cycles

The motion cycles considered here are cyclic walk or run cycles[1], where the character is walking on the spot with a would-be moving ground beneath with a constant velocity. The character is assumed to be walking on a flat horizontal ground and in the motion cycle each leg of the character takes exactly one step.

The choice was made to assume a horizontal ground in the implemented system. This assumption makes the analysis simpler. With some more sophisticated analysis methods than the ones presented here, motions on surfaces of any slopes could be analyzed, and the slope for each motion could also be automatically detected. However, supplying example motions for surfaces of different slopes could quickly exponentially increase the amount of example motions. While using a large amount of example motions as input for the generated motion as runtime is not directly opposed to the approach taken in this thesis, it is not the primary focus either, so for the simplicity of implementation, motions on non-horizontal surfaces have been disregarded here. The same applies to motion with turning, where a character is turning around while walking. This too could be analyzed with a little extra sophistication, but is not covered in the described methods and in the implemented system.

Time is defined cyclically for the motion cycles, with a value of 0.0 denoting the start of the cycle and 1.0 denoting the end, after which the cycle starts over again. The motion cycles are based on key-framed or prepared motion captured animations. There is no convention for which part of a motion should be the start, so this is effectively arbitrary. For a humanoid character this could for example be

[1] No distinction is made between walk and run cycles; to my knowledge there is simply no general term that covers both.



(a) *Leg cycles relative to motion cycle. This example shows two legs with stance times opposite each other in the motion cycle.*

(b) *Keytimes in leg cycle. Each leg has a leg cycle with the leg cycle time going from 0.0 at the stance time to 1.0 at the stance time again.*

Figure 4.1: *Motion cycle and leg cycles.*

a state where the left foot is down, or the right foot, or where both feet are touching the ground, but nothing can be assumed.

There can be any number of legs and the movements of the different legs are analyzed completely independently from each other. For each leg a heuristic is used to find a *stance time*:

Stance time: The point in time in the motion cycle when the leg is standing most firmly and neutrally on the ground.

In addition to the overall motion cycle, each leg has its own leg cycle. The start and end of these leg cycles are marked by the stance times of the respective legs, so contrary to the overall motion cycle, the start times of the leg cycles are not arbitrary at all. The relation between the leg cycles and the motion cycle can be seen in figure 4.1a.

4.2.1 Keytimes in Leg Cycles

In order for a procedural animation method to properly handle the provided example motions, it may need to know certain properties of those motions. Keytimes signify the times of certain important events in the motion, for example at which times in the cycle each foot lifts off the ground.

Rather than using keytimes called *toe-off* and *heel-strike* to mark the times when a foot lifts off the ground and strikes the ground again, respectively, a different but similar set of keytimes is used in this work. Since the methods described here are not meant to be used exclusively for humanoid characters, the concepts have been generalized to only make a minimal number of assumptions about the anatomy and the style of locomotion of the character in question. Specifically, the system described here does assume that two points, roughly opposite each other, on the under-side of the foot can be identified, and they will be referred to in the following as the heel

and toe points for convenience. However, the system makes no assumptions about which of these two points first touch the ground and leave the ground in a walk cycle. Thus, toe-off is generalized to *foot-off* and heel-strike time is generalized to *foot-strike*. The foot-off, foot-strike, and all the other leg cycle keytimes are relative to the leg cycle time that spans from 0 to 1 (see figure 4.1b).

Foot-off: The time in the leg cycle when the foot lifts off the ground entirely.

Foot-strike: The time in the leg cycle when the foot first strikes the ground.

Two additional keytimes are used to signify the span of time when the foot stands almost flat on the ground:

Foot-lift: The time in the leg cycle when some part of the foot is beginning to lift so that the foot is no longer standing flat on the ground.

Foot-land: The time in the leg cycle when the foot is again standing flat on the ground.

The time-span between foot-strike and foot-land is used to determine the duration of the foot-roll when landing on the ground. However, if the heel and toe touch the ground simultaneously when landing, the foot-strike and foot-land time can be the same. To avoid a foot-roll of too short duration, an additional keytime is used called *pre-foot-land*. Normally the pre-foot-land time equals the foot-strike time, but if the time-span between foot-strike and foot-land is smaller than a given threshold, the pre-foot-land time is instead the foot-land time minus that threshold. Similarly, a keytime called *post-foot-lift* is used that is greater than or equal to the foot-off time.

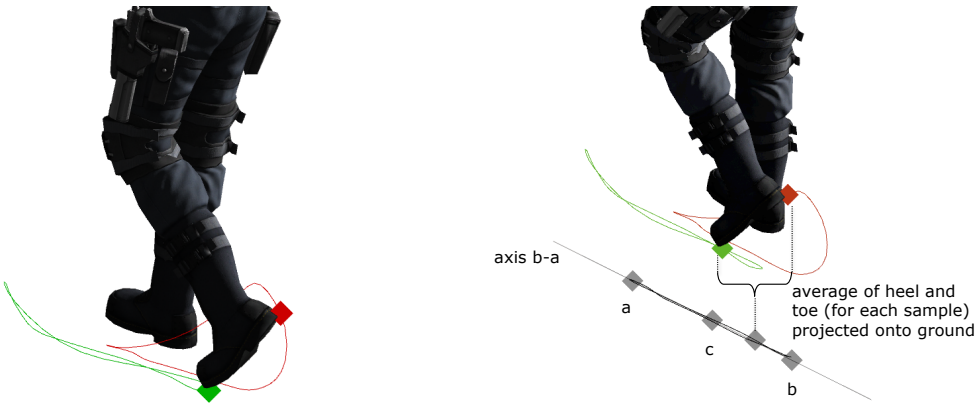
Post-foot-lift: Equals foot-off or foot-lift + minimum desired foot-roll duration, whichever one is greater.

Pre-foot-land: Equals foot-strike or foot-land - minimum desired foot-roll duration, whichever one is smaller.

For each leg the stance time and all the leg cycle keytimes are analyzed without user intervention as part of the motion analysis.

4.3 Foot Movement Analysis

The first step in the motion analysis is for each leg to determine the stance time and the key times in the leg cycle, the footbase trajectory, and the direction and length of the foot step. First the trajectories of the heel and toe are sampled throughout the motion cycle (see figure 4.2a) and all the other information is derived from that.



(a) *The trajectories of the heel and toe of the foot are obtained by sampling the heel and toe positions throughout the motion cycle.*

(b) *An axis of movement* $\mathbf{axis} = \mathbf{b} - \mathbf{a}$ *is estimated based on trajectory of average of heel and toe points projected onto the ground. The average point* $\mathbf{c}$ *is calculated;* $\mathbf{a}$ *is the point furthest away from* $\mathbf{c}$*, and* $\mathbf{b}$ *is the point furthest away from* $\mathbf{a}$*.*

Figure 4.2: *Heel and toe trajectories and estimation of an axis of movement.*

4.3.1 Movement Axis

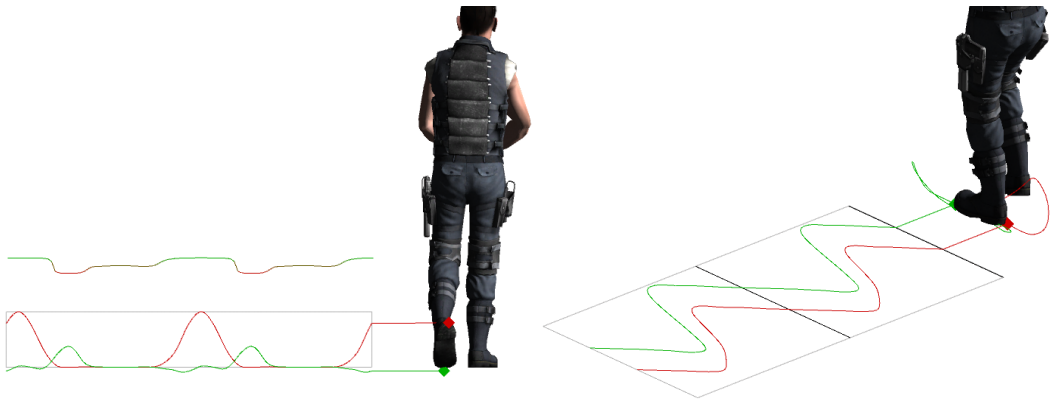
Given the heel and toe trajectories, an estimate of the axis of movement can be calculated. An average of the heel trajectory and toe trajectory is calculated by taking the average of the heel and toe position for each sample s:

$$\mathbf{middle}_s = \frac{\mathbf{heel}_s + \mathbf{toe}_s}{2} \tag{4.1}$$

This trajectory is then projected onto the ground plane. A point $\mathbf{c}$ is calculated that is the average of all the points in the projected trajectory. A point $\mathbf{a}$ in the projected trajectory is identified that is the point furthest away from $\mathbf{c}$, and a point $\mathbf{b}$ is identified in the projected trajectory that is the point furthest away from $\mathbf{a}$. Now the vector $\mathbf{axis} = \mathbf{b} - \mathbf{a}$ is a good estimate of the axis of movement. See figure 4.2b. At this point it is still not known in which direction along the axis the character moves, or with which speed.

4.3.2 Stance time

With an estimate of the axis of movement given, the swinging of the leg along that axis can be measured, and a heuristic can be used to find the *stance time*, which is the point in time in the motion cycle when the leg is standing most firmly and neutrally on the ground. To determine the stance time, a cost value is calculated for each sample s based on the height of the heel and toe at that sample (refer to figure 4.3a) as well as the position of the foot middle (as defined in equation 4.1) along the axis of movement (see figure 4.3b). The time of the sample with the lowest cost



(a) *Curves showing heel and toe heights throughout the leg cycle. Above these a balance curve is shown. The balance value signifies which part of the foot is closer to the ground at a given time and thus should act as pivot point when rotating the foot.*

(b) *Swing of heel and toe along estimated axis of movement, with vertical black lines in the graph indicating the stance time in the leg cycle. The character is portrayed here frozen at the stance time of the left leg: The point in the cycle where this leg is standing most firmly and neutrally on the ground.*

Figure 4.3: *Movements of heel and toe along the vertical axis and along the estimated axis of movement of the character.*

is used:

$$cost_s = \frac{\max(heelheight_s, toeheight_s)}{\alpha} + \frac{\left|(\mathbf{middle}_s - \mathbf{c}) \cdot \hat{\mathbf{axis}}\right|}{\beta} \tag{4.2}$$

where α and β are parameters that can be tweaked to control whether it is more important that the foot at the stance time is as low as possible or as close to the middle of the swing as possible. In the implemented system α was set to the highest observed heel or toe height across all the samples, and β was set to $\|\mathbf{b} - \mathbf{a}\|$.

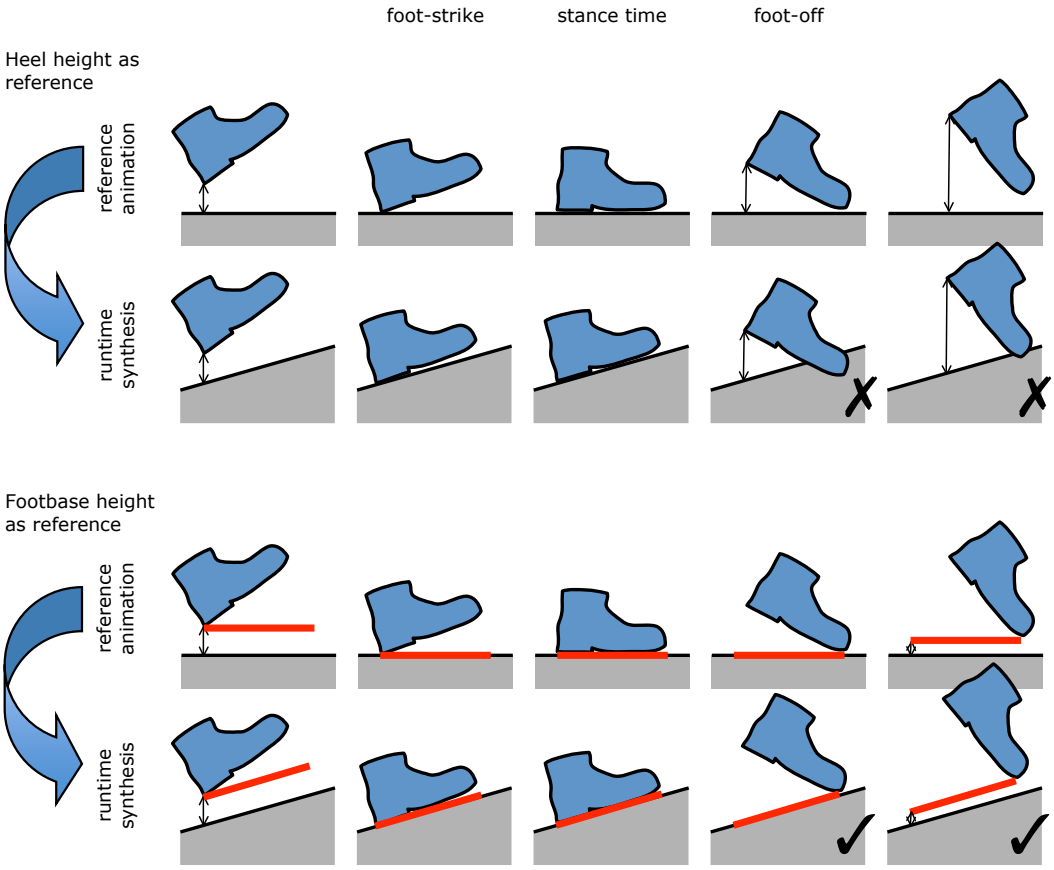
Given the stance time, the *foot direction* at that time in the cycle is stored. The foot direction is the vector $\mathbf{toe} - \mathbf{heel}$ projected onto the ground plane, normalized, and then multiplied with the length of the foot:

$$\mathbf{footdirection} = \hat{\mathbf{d}} \cdot footlength \tag{4.3}$$

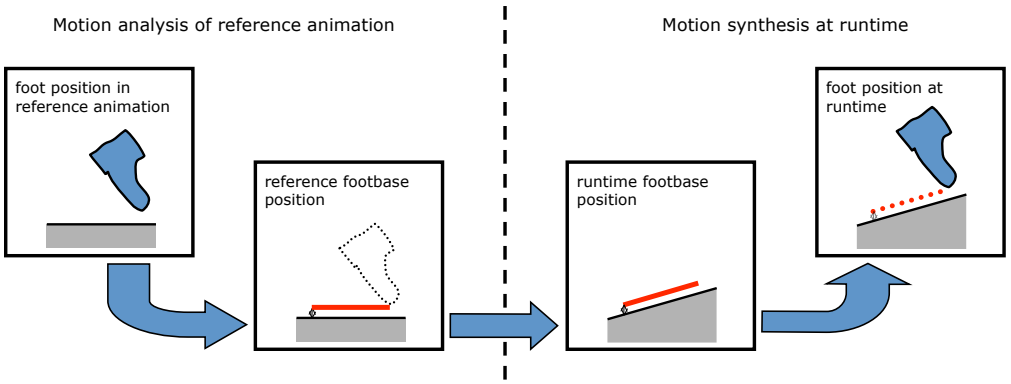
$$\mathbf{d} = (\mathbf{toe} - \mathbf{heel}) \parallel \mathbf{ground} \tag{4.4}$$

4.3.3 Balance and Footbase

When procedurally adjusting the movements of the feet at runtime, the alignments of the feet relative to the ground may not always match the corresponding alignment in the original input animation. The foot alignment will often need to be procedurally adjusted when walking on a sloped surface or to create a proper foot-roll when taking a step up or down a step. In these cases it is important to consider which part



(a) *Foot alignment comparison. At the top is depicted how using the heel height as reference can produce undesirable results in the procedural runtime motion synthesis. Using the toe height as reference will have similar problems. To solve these problems, a footbase is used as reference.*



(b) *The reference footbase position is derived from the position of the foot in the reference animation. At runtime the position of the foot is based on the runtime footbase position.*

Figure 4.4: *The* footbase *is a line segment that has the same length as the foot and which is always parallel to the ground below the foot. Depending on whether the heel or toe is closer to the ground, the foot touches the footbase with either the heel or toe, respectively.*

of the foot is used as the pivot point that the foot is rotated around. To always get correct results, the rotation should always be around whichever part of the foot is closer to the ground. See figure 4.4a. To achieve this, I introduce the concept of a *footbase*. The footbase is a line segment that has the same length as the foot and which is always parallel to the ground below the foot. Depending on whether the heel or toe is closer to the ground, the foot touches the footbase with either the heel or toe, respectively.

To get a measurement of which part of the foot is lower (and thus closer to the ground) at a given time, a *balance* value is calculated for each sample *s*. A balance value close to zero indicates that the heel is lower and a value close to one indicates that the toe is lower (see figure 4.3a). The balance for sample *s* is calculated with

$$balance_s = \frac{\arctan\left(\frac{heelheight_s - toeheight_s}{footlength}\alpha\right)}{\pi} + 0.5 \tag{4.5}$$

with α denoting a sensitivity parameter, where $\alpha < 1$ creates a bias towards the middle value of 0.5 while $\alpha > 1$ creates a bias towards the extremes of 0 and 1. In the implemented system the value $\alpha = 20$ was used, since a high bias towards 0 and 1 was desired.

Given the *balance* value for a sample *s*, the position of the *footbase* of the foot can be derived. The end of the footbase corresponding to the heel is defined as its position. The direction of the footbase along the ground is based on the *foot direction*, as given in equation 4.3.

$$\mathbf{footbase}_s = \mathbf{heelpos}_s \cdot balance_s + (\mathbf{toepos}_s - \mathbf{footdirection}) \cdot (1 - balance_s) \tag{4.6}$$

The reference footbase position is dictated by the position and alignment of the foot, as seen above. At runtime the footbase is continuously adjusted to be parallel to the ground below the foot and the position of the foot is dictated by the alignment of the foot and the position of the footbase (see figure 4.4b). This will be covered in more detail in chapter 7.

The footbase position for each sample is stored and the one for the sample associated with the *stance time* is referred to as the *stance position*:

Stance position: Position of the *footbase* relative to the body at the *stance time*, where the foot is standing most firmly and neutrally on the ground.

4.3.4 Keytimes

Knowing the stance time and using the trajectories of the heel, toe, and footbase, the keytimes (see sub-section 4.2.1) of the leg cycle can be found.

Figure 4.5 shows an example of an analysis, but note that the details are highly dependent on the character and on the walking or running style. Several properties of the trajectories are analyzed to find the keytimes.

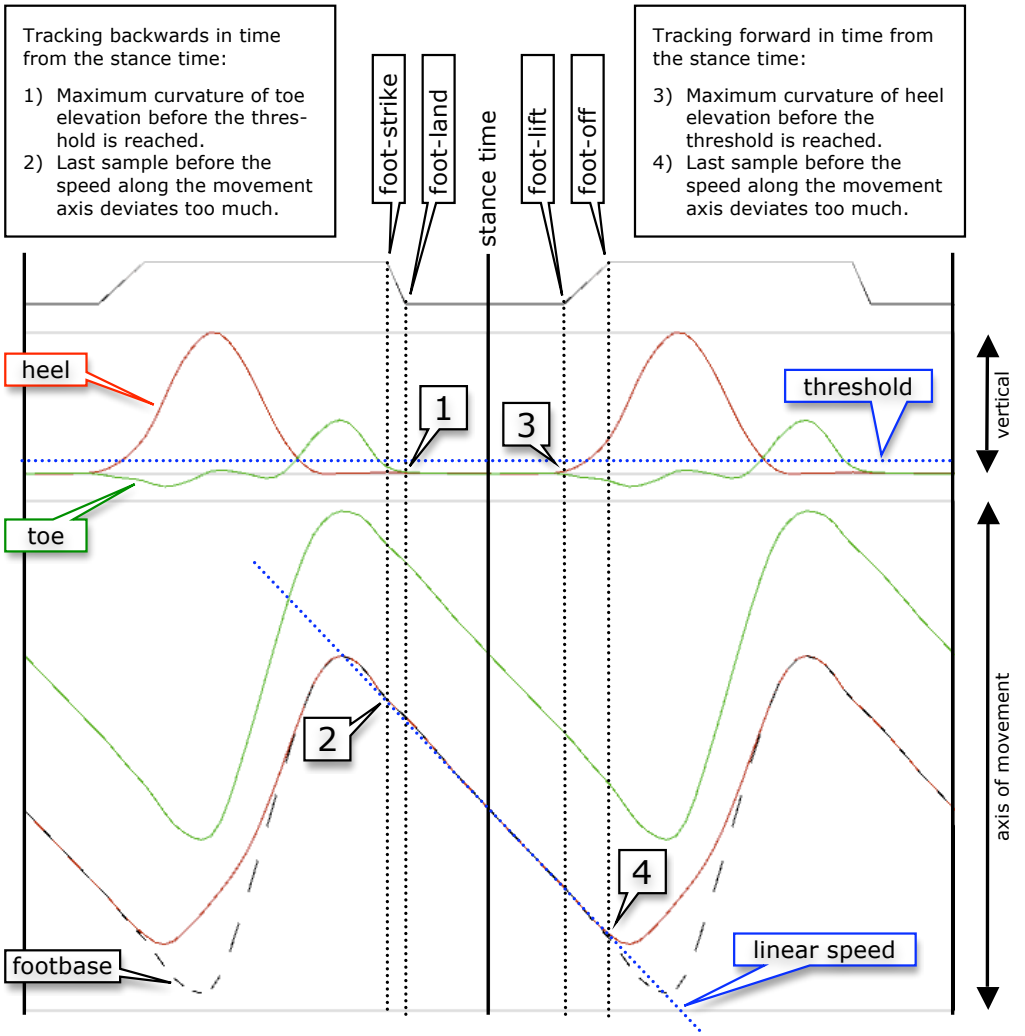


Figure 4.5: *Example of calculation of keytimes.*

- The vertical component of the trajectory of the heel is tracked from the stance time and forward. The time of the sample where the heel trajectory has its maximum curvature before the heel reaches a certain threshold altitude is stored temporarily as the *heel-off* time.
- The same tracking is performed for the vertical component of the trajectory of the toe to find the *toe-off* time, which is also stored temporarily.
- The horizontal component of the footbase is tracked from the stance time and forward. Since the foot is resting on the ground at the stance time, and the ground is moving with a constant speed relative to the character, the footbase is moving with a constant speed for at least some duration before and after the stance time. This speed is calculated. The point in time when the horizontal movement of the footbase deviates from this constant speed too much is stored temporarily as the *linear-speed-end* time.

Given these temporary time values, the *foot-lift*, *foot-off*, and *post-foot-lift* keytimes are calculated:

$$\text{foot-lift} = \min(\text{heel-off}, \text{toe-off}, \text{linear-speed-end}) \quad (4.7)$$

$$\text{foot-off} = \max(\text{heel-off}, \text{toe-off}) \quad (4.8)$$

$$\text{post-foot-lift} = \max(\text{foot-lift} + \alpha, \text{foot-off}) \quad (4.9)$$

where α is the desired minimum duration of a foot-roll expressed as a fraction relative to the entire duration of one cycle. In the implemented system $\alpha = 0.2$ was used.

The other three keytimes are calculated in a completely analogous fashion, except in reverse. Temporary time values *heel-strike*, *toe-strike*, and *linear-speed-begin* are found by tracking the trajectories from the stance time and backwards rather than forward. Given these temporary time values, the *foot-land*, *foot-strike*, and *pre-foot-land* keytimes are calculated:

$$\text{foot-land} = \max(\text{heel-strike}, \text{toe-strike}, \text{linear-speed-begin}) \quad (4.10)$$

$$\text{foot-strike} = \min(\text{heel-strike}, \text{toe-strike}) \quad (4.11)$$

$$\text{pre-foot-land} = \min(\text{foot-land} - \alpha, \text{foot-strike}) \quad (4.12)$$

The six keytimes are stored permanently and used both in the further analysis and at runtime.

4.3.5 Foot Step Direction and Length

With the keytimes of the leg cycle known, the direction and length of the stride can be calculated trivially by measuring the distance the footbase has moved (in character space) between *foot-strike* and *foot-off*, and divide the length of the vector by the time span between the two keytimes:

$$\mathbf{displacementvector} = \mathbf{footbase}_{footoff} - \mathbf{footbase}_{footstrike} \quad (4.13)$$

$$stridelength = \frac{\|\mathbf{displacementvector}\|}{footoff - footstrike} \quad (4.14)$$

$$\mathbf{stridedirection} = -\hat{\mathbf{displacement}}\mathbf{vector} \quad (4.15)$$

The stride direction is the negative direction of the displacement vector. For example, if the foot moved backwards in character space from *foot-strike* to *foot-off*, it means that the character is walking forwards.

4.4 Determining Motion Cycle Properties

With the stride length and direction known for each leg, the overall direction and distance of the motion cycle is found by simple average, and the speed is found by dividing the distance by the duration of the motion cycle:

$$cycledistance = \frac{\sum_{l=1}^{legs} stridelength_l}{legs} \quad (4.16)$$

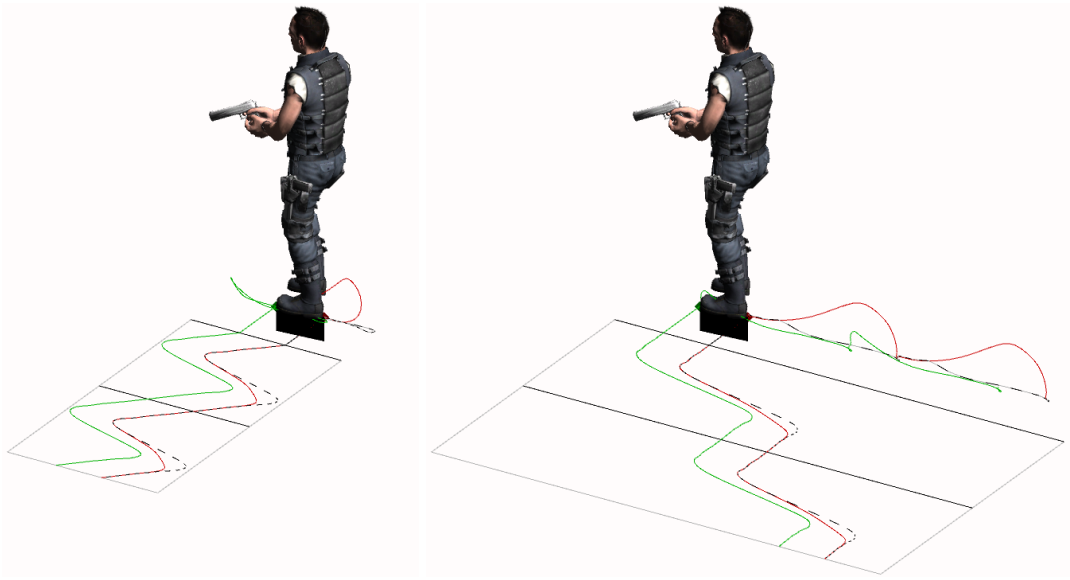
$$\mathbf{cycledirection} = \frac{\sum_{l=1}^{legs} \mathbf{stridedirection}_l}{legs} \quad (4.17)$$

$$cyclespeed = \frac{cycledistance}{cycleduration} \quad (4.18)$$

While the average is not very sophisticated, it works well in practice. For motions of high quality, the analyzed length and direction of the stride of each leg should be identical anyway. Theoretically, the strides of the different legs can only differ if the character does not keep a constant velocity throughout the motion cycle, and it is an assumption in this work that all motion cycles have a constant velocity. In the case of different stride lengths or directions caused by imprecision of the automated analysis or by the example motions being of poor quality, the average ensures that any error is spread out among the legs so as to be minimally visible. For example, if a provided motion cycle has one leg taking a longer step than the other, the result at runtime will be that one foot slides slightly backwards while on the ground, while the other foot slides slightly forwards. This assumes that the motion is used as-is and no measures are taken to enforce that the feet stay fixed while on the ground. In the implemented system such a feature can optionally be turned on.

4.4.1 Normalized Footbase Trajectories

At this point the trajectory of the footbase for each foot is known in character space. In order to be used easily at runtime, the footbase trajectories must be transformed into a normalized form, where the stride always goes along the z axis from the position $(0,0,0)$ to $(0,0,1)$. First, the trajectories in global space are found by first



(a) *Trajectories in character space.* **(b)** *Trajectories in global space.*

Figure 4.6: *Trajectories for heel (red), toe (green), and footbase (dotted black) in character space, and global space.*

tracing the footbase (still in character space) relative to a reference point that moves in the reverse cycledirection by the cycledistance over the duration of the cycle (see figure 4.6). This trajectory is then rotated by a rotation Q such that is always is along the z axis.

$$cycletime_s = time_s - stancetime_s \tag{4.19}$$

$$\mathbf{reference}_s = \mathbf{stanceposition} - (cycledistance \cdot \mathbf{cycledirection} \cdot cycletime_s) \tag{4.20}$$

$$\mathbf{normalizedfootbase}_s = Q \cdot (\mathbf{footbase}_s - \mathbf{reference}_s) \tag{4.21}$$

where Q is the rotation required to rotate the vector **cycledirection** into the vector $(0,0,1)$. After this, the z component of the **normalizedfootbase** is divided by the *cycledistance* in order to make the normalized stride always have a length of 1.

4.5 Results

The described methods for motion analysis have been used on a variety of characters and motions (see chapter 8 for details). The most challenging part of the analysis is the identification of the correct keytimes. While the described heuristics do a good job in general of finding keytimes that are precise enough to produce good results when used by the semi-procedural animation methods at runtime, the identified keytimes occasionally appear, when closely inspected, to be a few frames off the time that a human would have chosen as the right moment. The most deciding factor that has influence on the correct analysis of the keytimes is the exact method used for

calculating the curvature of the heel and toe trajectory curves. This method decides when the heel and toe are identified to lift off the ground and land on the ground. There is a question of which filtering to apply before the curvature is measured, in order to exclude small curvature changes caused by noise or inaccuracy, and there is a question of how best to ensure that curvature changes where the heel or toe are still close to the ground are given preference. These details in the implemented system are currently based on trial and error. Further research into the issue could perform more rigorous tests where a selections of characters and animations are all tested with different methods and values for variables, and the results inspected and graded in a structured manner to decide which algorithms and variable values produce the most consistent high quality identification of keytimes.

The successful automated motion analysis is based on certain assumptions about the example motions used as input. In order to be analyzed with little or no error by the method, the given motions must live up to a few prerequisites:

- The motion should be cyclic and over the duration of the cycle each leg should take exactly one step.
- The feet of a character should be at their lowest when resting on the ground. In other words, the feet should preferably never be intersecting the ground. While this might seem like something that could be taken for granted, certain tested motions of less than ideal quality exhibited noticeable intersection with the ground during certain parts of the motion cycle. The analysis method can tolerate a fair amount of such flaws in the analyzed motion, but with increasing chances of errors in the analysis.
- When in contact with the ground, feet should be moving with constant velocity relative to the character. Furthermore, all the feet should be moving with the same velocity. For example, for a character walking forward with a certain speed, the character's feet should all be moving backwards with that same speed while they are in contact with the ground. Motions that do not live up to this requirement would exhibit footskate in their original, unadjusted state, and the automated analysis methods are based on an assumption that the example animation do not have such footskate. Again, the method can tolerate a certain amount of error on this part, but the more inconsistent velocities, the higher the risk of errors in the analysis.

The prerequisites are normally easily met by walk or run cycles that have been made from cleaned-up motion-capture or manually by skilled animators. However, walk or run cycles of lower quality, that suffer from visible artifacts from the beginning, do not always live up to the prerequisites.

4.6 Summary of Motion Analysis

Walk and run cycles are cyclic motions in character space, where the character is walking on the spot with a would-be moving ground beneath with a constant

velocity. The character is assumed to be walking on a flat horizontal ground and in the motion cycle each leg of the character takes exactly one step. For each leg a a number of properties are stored with regard to how that leg takes a step. The analysis is primarily based on observations of the trajectories of the heel and toe, and information that can be derived from this. A leg cycle is defined for the leg, which a number of keytimes specifying the times in the leg cycle when special events occur, such as the leg standing firmly on the ground (*stance-time*), taking off the ground (*foot-lift*, *foot-off*, and *post-foot-lift*), and landing on the ground again (*pre-foot-land*, *foot-strike*, and *foot-land*). Having found these, the distance and velocity of the motion can be calculated, and the trajectory of each foot is also stored in a special normalized form, which makes it simple to later blend trajectories from multiple motions together at runtime.

Chapter 5

Motion Blending

Blending enables the bones of a character to not just be moved by a single example motion, but rather be moved simultaneously by multiple motions, using a weighted average. This chapter discusses the approach taken in this thesis to providing high-level control of stylistic motion parameters while keeping the control of physical motion parameters fully automated. Additionally, methods are presented for synchronizing motions prior to blending by means of cycle alignment and motion warping.

5.1 Controlling Motion Parameters and Transitions

Example motions can vary in multiple respects. Speed and direction are two dimensions in which example motions can have different parameters. While speed and direction are *physical parameters* that can be measured, other parameters, such as happiness, briskness, and other *stylistic parameters* relating to style, personality, and emotion, cannot be measured in an automated fashion. These parameters can however be identified and annotated manually by artists or designers. With a large number of example motions with different physical and stylistic parameters, the task of assigning proper weights to a subset of the example motions can become quite complex. Automated methods can take care of certain parts of this task and provide a higher level control scheme to the user that makes it simpler to achieve a blended motion with a desired target parameter.

5.1.1 Related work

Unuma et al. (1995) has used Fourier techniques to interpolate and extrapolate motion data for human articulated characters, including stylistic properties such as "tiredness", and "briskness". The method produced a realistic transition from walking to running, while the reverse transition from running to walking by the method was unnatural (Unuma et al., 1995, p95).

Rose et al. (1998) has proposed a framework of *verb and adverbs* to control the parameters (in the work called *adverbs*) of motions and the transitions between

distinctively different actions (in the work called *verbs*) such as walking versus jumping. In the system implemented by Rose et al. (1998) the physical parameters of turning angle and of walking uphill-downhill were used in the work, while artists annotated stylistic parameters such as happy-sad and knowledgeable-clueless (Rose et al., 1998, p34). The parameters form a parameter space, with a dimensionality equal to the number of parameters. To assign blend weights to the example motions within that parameter space at runtime, radial basis functions are used for scattered data interpolation. The interpolation technique itself has been improved upon by Sloan et al. (2001), while alternative interpolation techniques have been used in later works (Allen et al., 2002; Kovar and Gleicher, 2004), as will be expanded upon in the chapter 6 on Motion Interpolation.

Many works dealing with interpolation in parameter space simply do not address stylistic parameters at all and only work with physical parameters, assuming that example motions are similar in nature if they are measured to have physical parameters close to each other (Kovar and Gleicher, 2004; Srinivasan et al., 2005; Heck and Gleicher, 2007). This can work well in a computer game as long as there are no stylistic parameters that need to be controllable in the game, or if the implementation of the game can have other, orthogonal measures to control these aspects of motion.

Gleicher (2008) discusses the problem of managing interpolation between sample motions in a multidimensional parameter space as the demand for versatile motion increases. As the number of parameters grows, the space of possibilities grows exponentially, and this leads to increased demands on the number of examples required to adequately sample the space. At the same time, trade-offs between differences are difficult to compare. For example, if no immediately nearby example motion is available for a desired parameter vector, is it better to prioritize an example that is similar in tiredness, personality, or position (Gleicher, 2008, p89)?

An approach to constructing higher level blending parameter control that is widely used in the game industry, but which seem to be largely ignored in academia, is the concept of blend trees. Blend trees are offered in tools such as NaturalMotion morpheme and Havok Behavior, to name a few. A blend tree is a hierarchical grouping of motions. Each group node specifies the relative blend weights of the child nodes, which may be motions or sub-groups. The actual blend weight for a motion can then be calculated simply by multiplying the relative blend weights of all its parents up to the root.

5.1.2 Animation Groups

This work presents a system that can automatically identify physical parameters in two dimensions: Speed and (horizontal) direction, which is expressed in the form of a 2D vector with components for forward-backwards motion and sideways motion. It is also assumed that the designer of a game may want any number of stylistic parameters to be controllable. Rather than explicitly constructing a parameter space made from all the physical and stylistic parameters, the proposed method circum-

vents the issue of the dimensionality of the stylistic parameters by letting the designer handle that aspect of the interpolation manually through the use of *animation groups*.

An animation group is similar to a group of motions in a blend tree, and is defined here as a collection of motions that may have different physical parameters but all share the same stylistic parameters. For example, a "happy" animation group might have a motion for standing still (idle) as well as motions for walking and running if different directions, all of them with a "happy" style. A different "sad" animation group could have similar motions, but all with a "sad" style. However, the different groups don't have to include motions with corresponding physical properties, or even the same amount of animations. The "sad" group might for example have no motions for running.

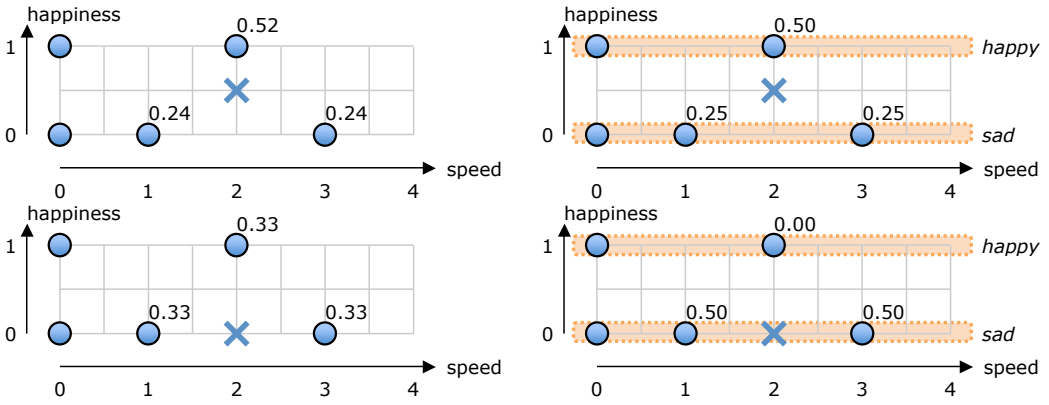
An overall blending weight is specified manually for each animation group. For example, the sad group could get a weight of 0.3 and the happy group a weight of 0.7. An arbitrary number of animation groups can have non-zero weights at the same time. Blending weights for the animations within a group are calculated automatically, as will be described in the chapter 6. They are then multiplied by the overall blending weight of the group. Blending techniques known from regular motion blending, such as cross-fading, can be applied to the animation groups as well. As such, they function as an abstraction layer that lets the designer only need to control the stylistic properties, while the automated method automatically pick the right blend weights to ensure that the right physical properties are met, which in the implemented system is speed and direction.

Compared to the use of a single multidimensional parameter space, this animation group approach has multiple advantages as well as some limitations. Animation groups effectively circumvent the issue of choosing trade-offs between differences in different parameters. The traditional interpolation in a single multidimensional parameter space can cause the blended motion to have unwanted stylistic properties (see figure 5.1a). The use of animation groups ensure that the intended stylistic weights are enforced in a simple and predictable fashion (see figure 5.1b). On the other hand, the use of animation groups means that example motions must belong entirely in a group. For example, with a "happy" and a "sad" animation group, example motions must fit in either one or the other, and cannot be in between, without explicitly creating a new animation group for that.

Essentially, the use of animation groups does not solve the problem of the number of required example motions growing exponentially with the number of controllable parameters. However, it does make it simpler to only supply animations for combinations that are actually needed and to get predictable blended motions when blending example motions with different physical and stylistic parameters.

5.2 Synchronization of Blended Motions

Before acceptable results can be obtained by blending multiple motions, the motions must be aligned in time such that similar events in the different motions occur at the



(a) *The blend weights for a desired parameter vector depends on how happiness is measured relative to speed. Top: Blend weights for halfway happy motion. Bottom: A happy example motion gets a third of the blend weight even though the desired parameter vector specifies zero happiness.*

(b) *Animation groups circumvents ambiguity of how stylistic parameters are measured relative to physical parameters. Top: Blend weights for halfway happy motion. Bottom: The happy example motion gets zero weight when the animation group to which it belongs have zero weight.*

Figure 5.1: *(a) Interpolation in a single multidimensional parameter space. (b) Interpolation first within and then between individual animation groups.*

same time across the motions. The example motions considered in this work are all cyclic walk or run cycles (with the special case idle animation being the exception, where no steps are taken at all). The primary method of synchronization used here is to align the different motion cycles such that the stance times of the different legs in the different cycles are as closely aligned as possible.

Besides the cycle alignment, time-warping is used to align the certain keytimes of the individual feet. Time-warping is the mapping of the actual time of a motion (e.g. a walk-cycle) onto generic time prior to blending, such that keytimes in the multiple motions are synchronized. To avoid artifacts when blending multiple motions that have their significant keytimes such as *foot-off* and *foot-strike* at different instants in time, time-warping is used to align the keytimes of the different motions prior to blending.

5.2.1 Related work

Rose et al. (1998) has presented a method for time-warping multiple motions to align their keytimes, by mapping the keytimes of the individual motions onto *generic time*. In generic time, the keytimes have the values 0, 1/n, 2/n, ..., 1, where n is the number of keytimes minus 1. At runtime, the keytimes of the blended motion are blended from the keytimes of the example motions with their respective blend weights. The motions are sampled in generic time and then “untimewarped” again to recapture phrasing and duration. This technique works well as long as the blend

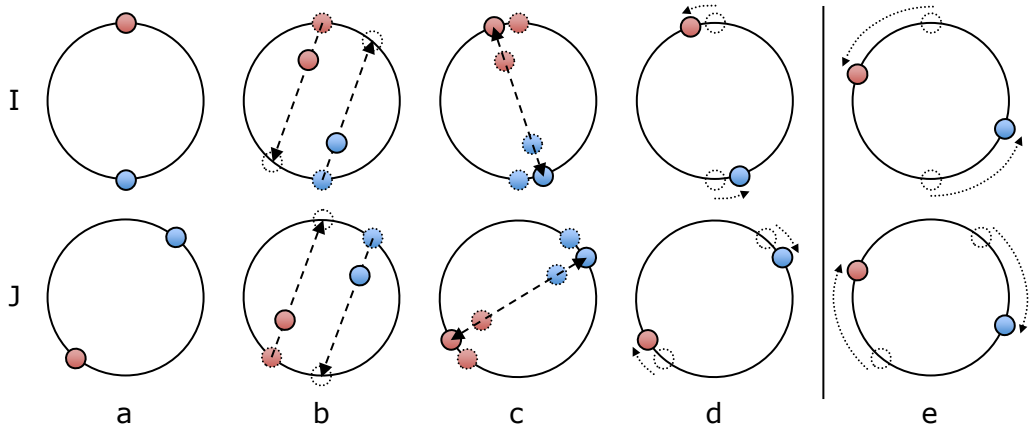


Figure 5.2: *Alignment of motion cycles. (a) The initial alignment of the stance time of the left and the right leg are different in motion cycle I and J. (b) The stance times are converted to 2D vector points and these are attracted towards the vector points of the corresponding stance times in the other motion. (c) The 2D vectors are converted back to time values. d: After one iteration, the stance times in the two motion cycles are closer aligned. (e) After multiple iterations the stance times are near-optimally aligned.*

weights remain the same throughout the motion. However, when that is not the case, a blended actual time may go reversely, back to the past, with respect to the generic time. Park et al. (2002) has presented an incremental scheme for time-warping that improves upon the method by Rose et al. (1998) in the general case where blend weights may change over the duration of a motion. This is done by blending the time-warping functions of the example motions incrementally for guaranteeing the monotonicity of the blended actual time. Specifically, the rates of change are blended rather than the actual times themselves (Park et al., 2002, p108). Kovar and Gleicher (2003) describe a way to automate the production of strictly increasing time-warp curves without user intervention. All the described methods rely on overall keytimes that apply to a character as a whole.

5.2.2 Cycle Alignment

A method is used in this work to align the different motion cycles such that the stance times of the different legs in the different cycles are as closely aligned as possible. This means that an animator does not have to worry about making different walk and run cycles match up. One walk cycle can begin with the left leg taking a step, and another with the right leg, and the system will automatically ensure that they are properly aligned prior to blending.

The alignment is stored as a time offset for each motion between 0 and 1, which is the same range that the time in a motion cycle is measured in. All cycle motions are compared to all the others during calculation of the offsets, which makes the computational complexity $O(n^2)$, where n is the number of motions. However, since the calculation is performed once at design time, and not at runtime, the computational cost is not a problem. The calculation is an iterative process based on

attracting the stance times of the corresponding legs in the different motion cycles, as illustrated in figure 5.2.

5.2.3 Time-Warped Normalized Footbase Trajectories

A normalized footbase trajectory is stored for each foot for each example motion (see sub-section 4.4.1 on page 37). Since the trajectories are normalized, they can be meaningfully averaged, and a weighted average is used with the same weights that are used for blending the example motions at the current moment.

Each example motion has its own set of keytimes. The *foot-off* time in one motion may be different from the *foot-off* time in another, etc. To address this, a time-warped look-up is made of those normalized footbase trajectories for all the motions that contribute to the current blended motion. The time-warping used here works individually on each leg. Using a single mapping based on the keytimes in all the legs would be nonsensical, given that the character may have any number of legs, and that the union of the keytimes for all the legs may not even occur in the same order in the different motions. The *generic time* per leg in this work has its first keytime at the *stance time*, the second at the *foot-off* time, the third at the *foot-strike* time, and the last at the *stance time* again to complete the cycle.

After the time-warped look-up of the normalized footbase trajectories has been performed on the relevant motions, a weighted average of those is performed based on the current blend weights. This time-warping technique is very simple and yet prevents undesirable artifacts that would otherwise have resulted from sampling curves from different motions that have their corresponding keytimes at different instants in time.

5.3 Results

While the use of animation groups have been shown to technically work, a lack of available motions has unfortunately made it impossible to perform any real testing of the approach in proper real-world scenarios. For example, no motions with similar velocities but different stylistic properties were available, meaning that testing of animation groups to control purely stylistic properties has not been tested in practice. Instead, tests have been conducted where walking motions were put in a separate animation group from running motions. As expected, when the running group is given full weight and the walking group zero weight, the character moves around in a running style of motion, no matter if the velocity matches the walking or running example motions best.

The cycle alignment algorithm works as expected in all cases where simple alignment of the cycles can produce a good match. This will always be true for motions of characters with two legs, where the legs take steps at times that are approximately opposite each other in the cycle. For characters with more than two legs, the algorithm also works well as long as the step order in the different motions can be aligned to approximately match. However, certain motions have step orders

that are different enough that no alignment exists that produce a good result (for example alignment between trot and gallop). Further research could look into possible general ways to produce natural transitions between inherently different gaits.

5.4 Summary of Motion Blending

Blending enables the bones of a character to not just be moved by a single example motion, but rather be moved simultaneously by multiple motions, using a weighted average. Automated methods can take care of certain parts of the blending weights assignment task and provide a higher level control scheme to the user. This thesis takes the approach of grouping sets of example motions into *animation groups,* that are collections of motions that may have different physical parameters but all share the same stylistic parameters. The weights of the individual motions within a group are automatically assigned while the weights of the animation groups are set by the user to control the *stylistic* properties of the motion.

Before multiple motions can be blended, they need to be properly synchronized. This thesis performs a simple cycle alignment to synchronize walk and run cycles prior to blending, and for each foot a simple form of time-warping is used when averaging the normalized footbase trajectories of the example motions that are blended.

Chapter 6

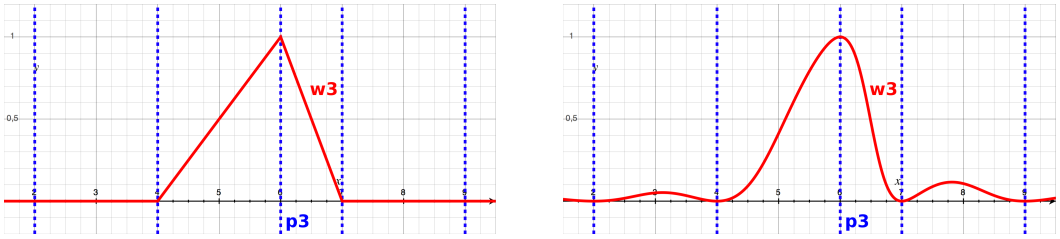
Motion Interpolation

Interpolation between several example motions is frequently a central element in generating flexible character animation in real-time. A number of example motions can be given, and their qualities be described in a parameter space. Possible parameters for walk cycles are speed and turning angle. In this work the parameters are the components of a velocity vector, where typically, one dimension measures forward-backwards speed and the other sideways speed. For a system supporting example motions with a character walking on a non-horizontal surface, a third parameter could measure a vertical component of velocity, though this is not supported in the implemented system at the time of writing. However, in general parameters can measure any desired property.

Let there be n example motions $\mathbf{M}_1, ..., \mathbf{M}_n$ with points in parameter space $\mathbf{p}_1, ..., \mathbf{p}_n$. The goal is to produce at any point $\mathbf{p}$ in the parameter space, a new motion $\mathbf{M}(\mathbf{p})$ derived through interpolation of the examples. When $\mathbf{p}$ is equal to the position $\mathbf{p}_i$ for a particular example i, then $\mathbf{M}(\mathbf{p}_i)$ should equal $\mathbf{M}_i$. In between the examples, smooth intuitive changes should take place.

The interpolated motion $\mathbf{M}(\mathbf{p})$ can be created as a linear combination of the example motions, using a set of blend weights $w_1(\mathbf{p}), ..., w_n(\mathbf{p})$ that specify the weight of each example motion at the point $\mathbf{p}$ in parameter space. What is thus needed is an appropriate weight function $w_i(\mathbf{p})$ for each example i.

Example in 1D Consider five walk cycles $\mathbf{M}_1, ..., \mathbf{M}_5$ with different speeds parameterized in a 1D parameter space, such that each motion has a point in parameter space corresponding to its speed: $\mathbf{p}_1 = 2, \mathbf{p}_2 = 4, \mathbf{p}_3 = 6, \mathbf{p}_4 = 7, \mathbf{p}_5 = 9$. Using interpolation it is then possible to obtain a walking motion with any speed $\mathbf{p}$ (with $2 \leq \mathbf{p} \leq 9$ when not using extrapolation). The weight function $w_3(\mathbf{p})$ corresponding to the example point $\mathbf{p}_3$ can be seen in figure 6.1 with two different interpolation methods and with $\mathbf{p}$ increasing along the horizontal axis.



(a) *Piecewise linear interpolation.*

(b) *Inverse distance weighting interpolation.*

Figure 6.1: *Illustration of weight function $w_3(\mathbf{p})$ in 1D, with (a) desired interpolation properties and (b) undesired interpolation properties for this work, since the function violates several of the desired interpolation constraints.*

6.1 Interpolation Constraints

For interpolation of motions it is desirable that the interpolation method conforms to certain constraints. Three constraints are described by Allen et al. (2002, p617) in the context of interpolating poses, and they are equally relevant for interpolation of motions. The three constraints are presented in a rephrased form here:

Exactness At an example point, the weight function for that example must evaluate to one, and all other weight functions must evaluate to zero. An exact fit to the example points is inherent in all interpolation methods. This is opposed to methods based on regression analysis, where an approximate best fit is estimated statistically. For this work an exact interpolation is desired due to the focus on preserving the original motions as intended by the animators.

Sum Since the weights are used for a weighted average of several motions, the weight functions must always sum to one.

Continuity The weight functions must be continuous so that animation will be smooth. Small changes in the parameter must result in small changes in the weights. Failure to meet this constraint will result in jitter in the blended motion. Only C^0 continuity is required.

Several interpolation methods enforce the *sum* constraint by *normalizing* the weights prior to blending, meaning that each of the weights are divided by the sum of all the weights. In the following, weight functions are denoted $w_i(\mathbf{p})$ and are assumed always to sum to one while weight functions prior to normalization, when used, will be called *influence functions* and denoted $h_i(\mathbf{p})$. The normalization is defined:

$$w_i(\mathbf{p}) = \frac{h_i(\mathbf{p})}{\sum_{j=1}^{n} h_j(\mathbf{p})} \tag{6.1}$$

Drawing from various prior research, the following four additional constraints have been formulated for this thesis and are considered desirable in addition to the three above:

Boundedness Blending with weights outside of the range from zero to one can give unpredictable and undesirable results (Allen et al., 2002, p617). Thus the weights should all be in the range from zero to one.

Locality Only example points close to the interpolated point should have positive weights, while all other weights should be zero. This ensures that only few motions have non-zero weights, which improves the efficiency of the blending. Furthermore, it is desirable that only motions with parameters close to the desired parameter have influence of the produced motion (Kovar and Gleicher, 2004, p566).

Monotony Each weight functions should decrease monotonically from its global maximum to its global minimum. Strictly speaking, the weight functions should have no local minima. For example, the weight of a running motion should decrease from one to zero as the speed decreases from running speed to walking speed. It is then undesirable if the weight of the running motion rises again to a positive weight when further decreasing the speed to even lower values. No prior work seem to formulate such a constraint explicitly.

Density-invariance Since this work aims to present a method suitable for use with potentially a very low number of example motions that may have arbitrary positions in parameter space, the interpolation method should work equally well with evenly and unevenly distributed example points and should not attribute greater weight to a nearby cluster of example points than to an equally nearby single example point. This attribute can also be called compensation for data density variation (Amidror, 2002, p173).

It can be seen in figure 6.1 that while inverse distance weighting interpolation violates the *locality* constraint and the *monotony* constraint, piecewise linear interpolation conforms to all the constraints. However, for this work a method is required that works for higher dimensions than 1D.

In the following sections a 2D example set of points will be used as a basis for comparisons of different interpolation methods. The example points in this set (see figure 6.2) are loosely based on the analyzed velocities from a set of animations used while developing the system.

6.2 Related Work

There is no obvious or naive generalization of piecewise linear interpolation to higher dimensions than 1D, but generally an interpolation method is desired in this work that degrades to piecewise linear interpolation in the 1D case, or have properties similar to this.

The problem of interpolating motions based on their respective positions in a multidimensional parameter space is essentially one of *scattered data interpolation*. A survey of often used scattered data interpolation methods for electronic imaging

Figure 6.2: *Velocities for a set of example motions.*

systems has been presented by Amidror (2002). While some of the methods assume interpolation of heights or other scalars or points, two of the described classes of methods are general enough to be applicable to the interpolation of motions, since they simply output a set of weights corresponding to each example point. These are inverse distance weighted interpolation and natural neighbor interpolation, which will both be examined below.

Also examined below are the methods of Cardinal Radial Basis Functions, K-Nearest-Neighbors, and Densely Sampled Parameter Space since these methods were developed specifically with the interpolation of motions or character poses in mind.

6.2.1 Inverse Distance Weighted Interpolation

Inverse distance weighted interpolation is a very simple interpolation that works in spaces of arbitrary dimensions and it is one of the most commonly used techniques for interpolation of scattered points (Amidror, 2002, p166). The method is global since it uses all the example points to calculate each interpolated value (Amidror, 2002, p166).

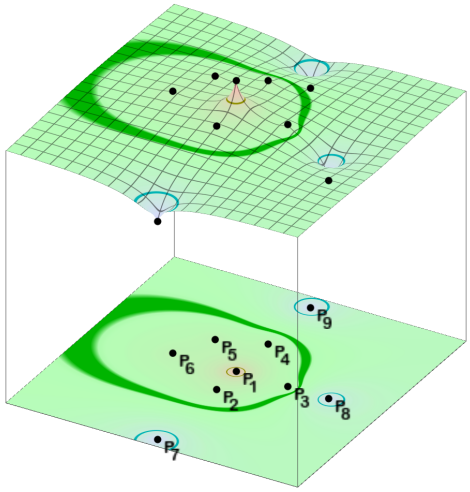
In inverse distance weighting interpolation the weight functions are as follows:

$$w_i(\mathbf{p}) = \frac{h_i(\mathbf{p})}{\sum_{j=1}^{n} h_j(\mathbf{p})} \tag{6.2}$$

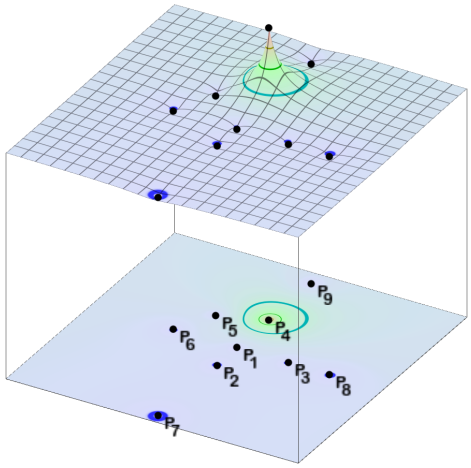
with influence functions:

$$h_i(\mathbf{p}) = \frac{1}{D(\mathbf{p},\mathbf{p}_i)^x} \tag{6.3}$$

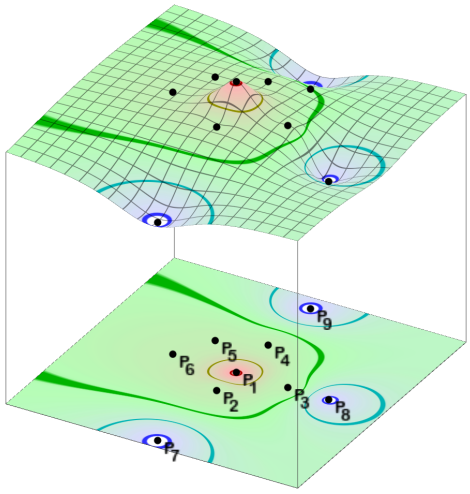
where $D(\mathbf{p},\mathbf{p}_i)$ is the distance between the new point and example point $\mathbf{p}_i$. The exponent x is typically set to 2 but can have other values as well. Figure 6.3 shows



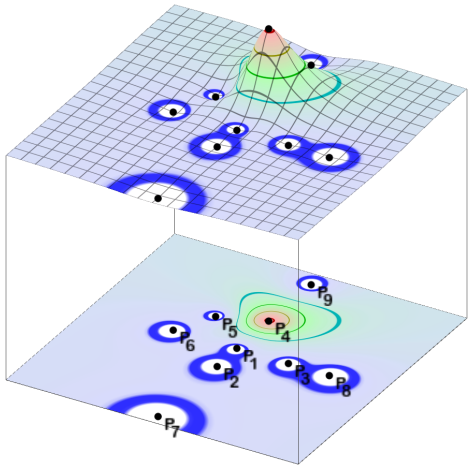
(a) *Interpolation of heights, with* $x = 1$.



(b) *Weight function* $w_4(\mathbf{p})$*, with* $x = 1$.



(c) *Interpolation of heights, with* $x = 2$.



(d) *Weight function* $w_4(\mathbf{p})$*, with* $x = 2$.

(e) *Color coding of graph heights.*

Figure 6.3: *Inverse distance weighting interpolation. (a,c) Interpolation with heights attributed to each example point. (b,d) Weight function* $w_4(\mathbf{p})$ *of example point* $\mathbf{p}_4$.

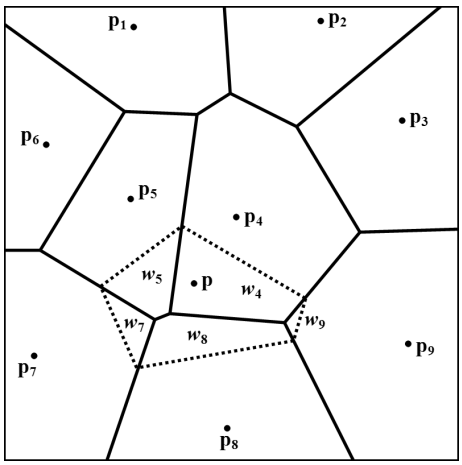


Figure 6.4: *Voronoi cells corresponding to each example point* $\mathbf{p}_1, ..., \mathbf{p}_9$ *and the new cell around the new point* $\mathbf{p}$*. The relative sizes of the areas "stolen" by the new cell from each of the existing cells produce the blend weights* $w_1, ..., w_9$*, though many of them do not have any area and are thus zero. Figure is derived from (Amidror, 2002, p172).*

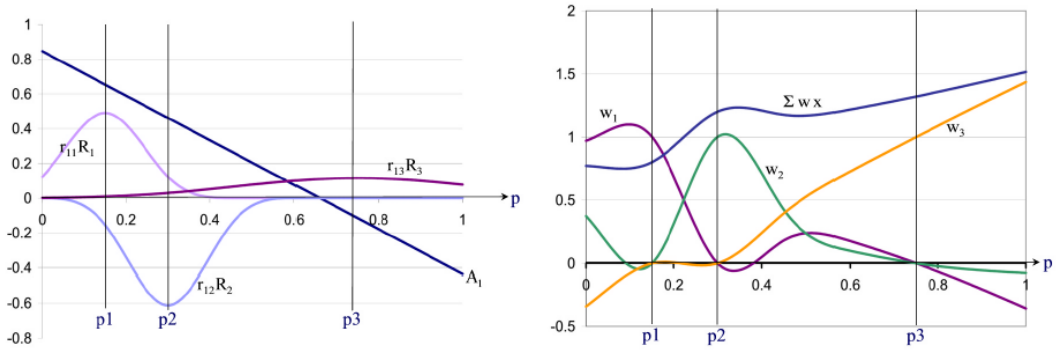
a set of example points interpolated with inverse distance weighting interpolation, with x set to 1 and 2. The method violates the constraints of *locality* and *monotony* discussed in section 6.1 on page 49 and is not *density-invariant*. While the method can be modified to be *local*, the other problems remain, and the method is not considered desirable for the purposes of this work. However, it is useful as a point of reference to compare other interpolation methods with.

6.2.2 Natural Neighbors Interpolation

An interpolation method with very desirable properties is natural neighbors interpolation (Amidror, 2002, p171). In natural neighbor interpolation the weights are area-based (or volume-based for higher dimensions), as opposed to distance-based like in many other methods. The parameter space is divided into a set of *voronoi* cells, with one cell corresponding to, and enclosing, each example point. The cell T_i is defined as the area of points that are closer to example point $\mathbf{p}_i$ than to any of the other examples (Amidror, 2002, p171):

$$T_i = \left\{ \mathbf{p} \in \mathbb{R}^2 \mid D(\mathbf{p}, \mathbf{p}_i) \leq D(\mathbf{p}, \mathbf{p}_j) \forall j = 1, ..., n \right\} \tag{6.4}$$

where $D(\mathbf{a}, \mathbf{b})$ denotes the Euclidean distance between the points $\mathbf{a}$ and $\mathbf{b}$. To evaluate the weights at a new point $\mathbf{p}$, the point is simply added to the set of example points, letting it "crave" its own cell from the cells of the surrounding example points (see figure 6.4), and the relative sizes of the areas "stolen" by the new cell from each of the existing cells produce the blend weights. Notably, this method is *density-invariant* due to the area-based weights. However, compared to many other methods, natural neighbor interpolation is quite complex to implement. Furthermore it is more computationally intensive than many other approaches, especially



(a) *Linear and radial parts of the cardinal basis function for the first example.*

(b) *Cardinal basis functions and scaled sum for particular degree of freedom.*

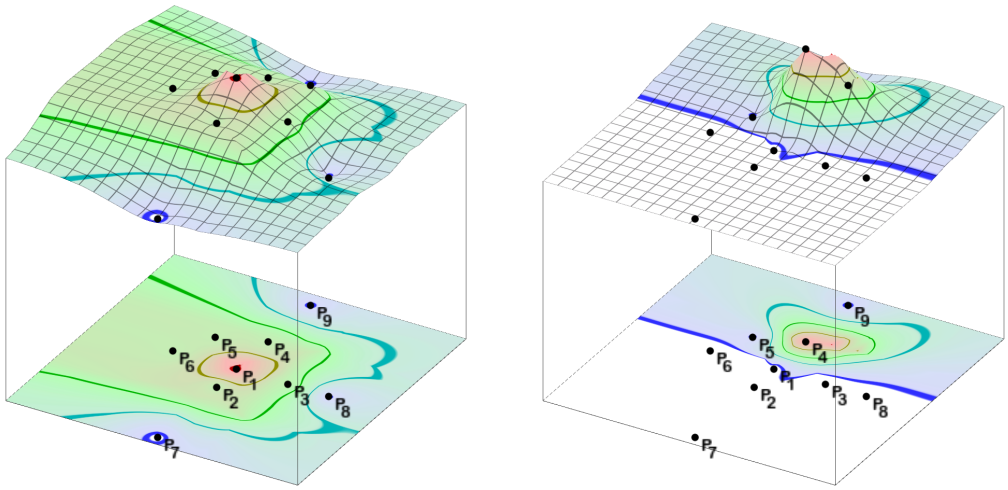
Figure 6.5: *The cardinal basis function w_1 in figure (b) is the sum of the linear and all the radial functions depicted in figure (a). Notice how w_1 is one at example $\mathbf{p}_1$ and zero at examples $\mathbf{p}_2$ and $\mathbf{p}_3$. In between and outside of examples, w_1 takes on arbitrary positive and negative values; as do w_2 and w_3. Figures are from Sloan et al. (2001, p138).*

for higher dimensions (Amidror, 2002, p173), and it has not been attempted to implement it for this work. It has however been used as a point of reference for evaluating other methods, particularly with regard to *density-invarianc*e.

6.2.3 Cardinal Radial Basis Functions

The technique of building parameterized motions from blends of captured examples was pioneered by Wiley and Hahn (1997) and the Verbs and Adverbs system of Rose et al. (1998). Cardinal radial basis functions is a multidimensional scattered data interpolation technique suggested by Sloan et al. (2001) as an improvement on the radial basis functions presented by Rose et al. (1998). Using cardinal basis functions, Sloan et al. improved the efficiency of the method by computing a weight for each example motion and performing blending based on those weights rather than interpolating each degree of freedom individually. Cardinal radial basis functions are used in several papers for interpolating poses and motions (Rose et al., 2001; Park et al., 2002; Semančík et al., 2004). The method assigns a function for each weight w_i that is the sum of a linear polynomial and a set of radial basis functions, with one radial basis function for each example.

The weight functions conform to the interpolation constraints of *exactness*, *sum*, and *continuity* as listed in section 6.1 on page 49. However, in between example points they take on negative weights (Allen et al., 2002, p617) and positive weights outside the range from zero to one (see figure 6.5) and thus violate the constraint of *boundedness*. Furthermore, if the user requests parameters far from the examples, blend weights are based purely on the linear approximation and hence are effectively arbitrary (Kovar and Gleicher, 2004, p561), further violating the constraints of *locality*, and *monotony*. Due to the radial basis functions that essentially give a radius of influence for each example point, the method is not *density-invariant*



(a) *Interpolation of heights, with* $k = 5$. **(b)** *Weight function* $w_4(\mathbf{p})$*, with* $k = 5$.

Figure 6.6: *K-nearest-neighbors interpolation.*

either, since an example point with a close neighbor in one direction and a neighbor further away in another direction cannot have a radius of influence that decreases to zero at the position of the close neighbor while also reaching all the way out to the other neighbor further away.

6.2.4 K-Nearest-Neighbors

K-nearest-neighbors interpolation is another method used in several papers for interpolating poses and motions, both directly (Allen et al., 2002) and as part of another interpolation method (Kovar and Gleicher, 2004; Heck and Gleicher, 2007). It relies on interpolating between the k nearest neighbor examples, with the influence functions (Allen et al., 2002, p617):

$$h_i(\mathbf{p}) = \frac{1}{D(\mathbf{p},\mathbf{p}_i)} - \frac{1}{D(\mathbf{p},\mathbf{p}_k)} \tag{6.5}$$

where $D(\mathbf{p},\mathbf{p}_i)$ is the distance between the new point $\mathbf{p}$ and example point $\mathbf{p}_i$, and where $\mathbf{p}_1,...,\mathbf{p}_k$ are the k nearest neighbors in order of increasing distance.

However, when $i = k$ then $D(\mathbf{p},\mathbf{p}_i) = D(\mathbf{p},\mathbf{p}_k)$ and this means that of the k nearest examples, the *kth* closest example will always have a weight of zero. Furthermore, examples that are equally far away as the *kth* closest example will also have a weight of zero. Thus, if the interpolated point $\mathbf{p}$ is equally far away from all the k nearest examples, then all the weights are zero, and the blending is undefined. Furthermore, the weights are not continuous around this undefined point. Thus, interpolation based on k-nearest neighbors violates the *continuity* constraint for interpolation listed in section 6.1. The discontinuities may not appear in practice, and rarely will for a value of $k = 12$ in 3D parameter space, as used by Allen

et al. (2002). The higher the chosen value of k, the less is the probability that any point is equidistant from all its k nearest example points. However, higher values of k also make the interpolation increasingly equivalent to inverse distance weighted interpolation with an exponent of 1, with all the problems that method entails.

Besides its proneness to *discontinuities*, k-nearest-neighbors interpolation is also lacking *density-invariance* and *monotony*, the latter especially for higher values of k.

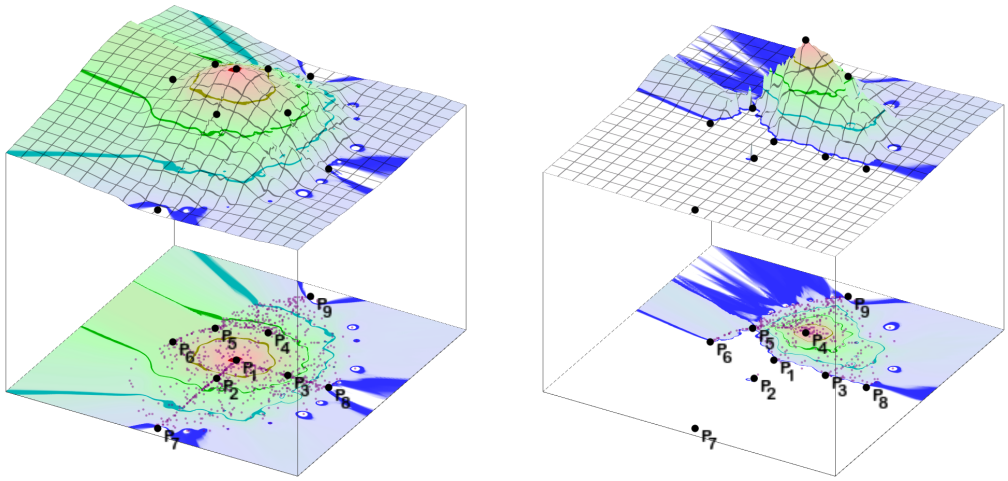
6.2.5 Densely Sampled Parameter Space

Many methods for building parameterized motions do not ensure that the actual parameters of motions synthesized by blending directly corresponds to the desired parameters. As pointed out by Kovar and Gleicher (2004, p561), this is reasonable for qualitative properties like happiness, where strict accuracy is not necessary (Unuma et al., 1995; Rose et al., 1998). Even when the blending is used to control measurable parameters, strict accuracy is unnecessary if the blending is not directly controlling those parameters. For example, Park et al. (2002) used blending for locomotion synthesis with specified speed and curvature, but the actual path that the character walked along was controlled by a specified trajectory. Similarly, in this work interpolation is used to control the weighting of different qualitative aspects of motions, such as the stylistic differences between walking and running, or between walking forward and walking sideways. However, after the motion blending, inverse kinematics (IK) is used to correct step length and foot placement. Thus, a high level of accuracy is not necessary from the parameters produced by the blending based on the interpolation.

Kovar and Gleicher (2004) describes a method for interpolating motions with blend weights that produce more accurate parameters. While the higher accuracy itself is not important for this work, the method is still interesting as an alternative form of interpolation that is strictly local and which overcomes some of the problems of pure k-nearest-neighbors interpolation, and because it has been widely adopted.

The basic idea is to densely sample the parameter space in order to obtain quick and accurate look-ups of blend weights. Since the problem is under-restrained, meaning that many combinations of blend weights can produce identical parameters, Kovar and Gleicher (2004, p566) suggest to limit blends to subsets of examples that are nearby in parameter space. For each sample a random point in parameter space within the bounding box of the example motions is considered, and the closest $d+1$ example motions are found, where d is the dimensionality of the parameter space. A random[1] blend of those $d+1$ examples is produced and its resulting parameter stored. At runtime a set of blend weights can be obtained for any position $\mathbf{p}$ in parameter space by performing k-nearest-neighbors interpolation on the sampled parameters (Kovar and Gleicher, 2004, p567) rather than on the original example

[1] Kovar and Gleicher (2004, p567) describes an algorithm for finding random blend weights for each of the $d+1$ examples, such that the weights sum to one.



(a) *Interpolation of heights.* **(b)** *Weight function* $w_4(\mathbf{p})$.

Figure 6.7: *Interpolation based on densely sampled parameter space; small gray dots are sampled parameters. In the weight function* $w_4(\mathbf{p})$ *quite visible artifacts can be seen between* $\mathbf{p}_5$ *and* $\mathbf{p}_6$ *resulting from random sampling based on the subset of example points* $\mathbf{p}_4$, $\mathbf{p}_5$, *and* $\mathbf{p}_6$.

motions.

While the method has been used successfully in a number of implementations where the interpolation weights are kept constant for each motion clip (Kovar and Gleicher, 2004; Heck and Gleicher, 2007), I have found that it is of limited use when dynamically and continuously changing the parameter position $\mathbf{p}$ over time since the method does not guarantee that nearby sampled parameters produce similar blend weights. The issue is best illustrated in the particularly severe case when a chosen subset of $d+1$ example points happen to be almost or completely co-linear. In that case, sampled parameters close to (or indeed positioned on) the center-most example point on that line may attribute anything from none of the weight to all of the weight to that example. When visualized in graphs, these artifacts in the weight functions appear as noise. Figure 6.7 shows a set of example points interpolated with densely sampled parameter space, as well as the weight function w_4 for example point $\mathbf{p}_4$.

The noise makes the weight functions highly *non-monotonous* and generally results in jerky motion when used to interpolate example motions with dynamically changing parameter values. A number of workarounds exist to deal with this[2], but none of them have been deemed desirable for this work.

[2]The noise of the weight functions can be countered by averaging a large number of the sampled parameters instead of using k-nearest-neighbor interpolation on them, but this makes the interpolation break the exactness constraint. Alternatively, non-desired elements could be excluded from the average, but that would make the interpolation non-deterministic, returning different blend weights at different times for the same parameter.

6.3 Gradient Band Interpolation

In this work a new interpolation method has been implemented that is particularly useful for interpolation in 2D or 3D space between a number of examples that each correspond to a specific velocity. The method does not ensure accuracy of the produced parameter values but it does ensure smooth interpolation under dynamically and continuously changing parameter values, and it more or less adheres to all the constraints listed in section 6.1, as discussed further down. The velocity for each example motion i is designated by the point $\mathbf{p}_i = (x_i, y_i, z_i)$ and the new velocity being queried is designated by $\mathbf{p} = (x, y, z)$. The interpolation specifies an influence function $h_i(\mathbf{p})$ associated with each example:

$$h_i(\mathbf{p}) = \min_{j=1}^{n} \left(1 - \frac{\overrightarrow{\mathbf{p}_i\mathbf{p}} \cdot \overrightarrow{\mathbf{p}_i\mathbf{p}_j}}{\left|\overrightarrow{\mathbf{p}_i\mathbf{p}_j}\right|^2} \right) \tag{6.6}$$

The influence function $h_i(\mathbf{p})$ of a example i creates gradient bands between the example point $\mathbf{p}_i$ and each of the other example points $\mathbf{p}_j$ with $j = 1...n, j \neq i$. Note that the fraction inside the parenthesis is identical to a standard vector projection of $\overrightarrow{\mathbf{p}_i\mathbf{p}}$ onto $\overrightarrow{\mathbf{p}_i\mathbf{p}_j}$, except for a missing multiplication with the vector $\overrightarrow{\mathbf{p}_i\mathbf{p}_j}$. The fraction instead evaluates to a scalar that is 0 when the would-be projected vector would have zero length and is 1 when the would-be projected vector would be equal to $\overrightarrow{\mathbf{p}_i\mathbf{p}_j}$. Thus, after this fraction is subtracted from one, each gradient band decrease from the value of 1 at $\mathbf{p}_i$ to 0 at $\mathbf{p}_j$ (see figure 6.8). The influence function evaluates to the minimum value of those gradient bands.[3]

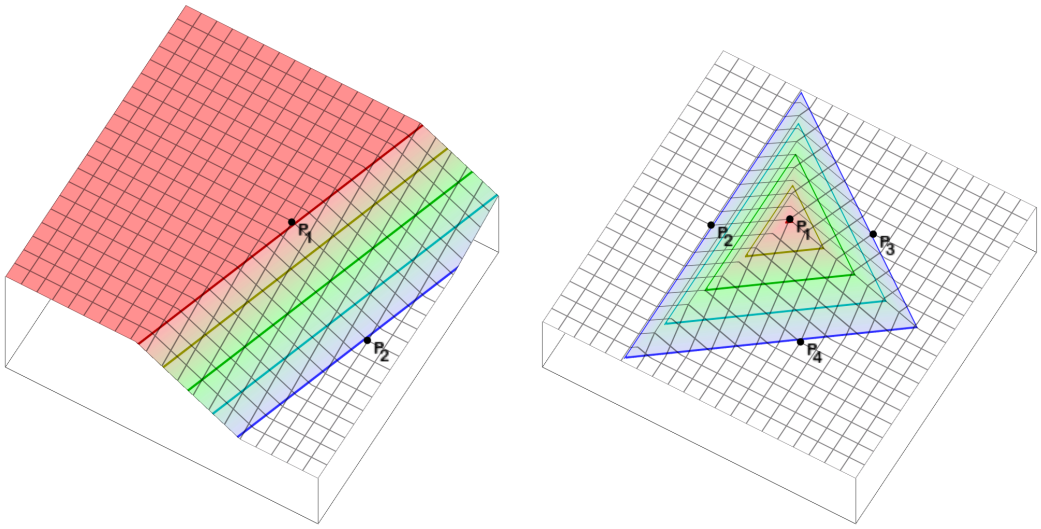
These influence functions are normalized to get the weight functions $w_i(\mathbf{p})$ associated with each example i:

$$w_i(\mathbf{p}) = \frac{h_i(\mathbf{p})}{\sum_{j=1}^{n} h_j(\mathbf{p})} \tag{6.7}$$

Due to the normalization the weight functions are not entirely identical to the influence functions: In some areas the influence functions may sum to more than one, and in those areas all the weights are divided by the sum (normalization). While the exact weights are altered by this, the overall shapes are quite similar and the extends of the weight functions (the areas in which they have non-zero weight) are not altered. See figure 6.9 for a comparison of influence and weight functions.

The gradient band interpolation conforms to all the constraints listed in section 6.1 on page 49. It is *exact*, since the gradient bands of $h_i(\mathbf{p})$ are always 1 at $\mathbf{p}_i$ and zero at all other example points. It *sums to one* due to the normalization, and it is *continuous* since the gradient bands are continuous, and the normalization does not introduce discontinuity. The weight functions are *bounded, local*, and

[3] Using the minimum of the gradient bands creates influence functions with hard edges. Various attempts have been made to get smoother influence functions, one of them being multiplying the gradient bands together. However, while the resulting functions are smooth, they have other more severe artifacts that make them unsuitable for interpolation, such as highly inconsistent steepness between examples.



(a) *Influence function $h_1(\mathbf{p})$. The gradient band decrease from a value of 1 at $\mathbf{p}_1$ to 0 at $\mathbf{p}_2$.*

(b) *Influence function $h_1(\mathbf{p})$. The function evaluates to the minimum of the three gradient bands.*

Figure 6.8: *Gradient band interpolation with (a) two and (b) four examples. Depicted here are the influence functions of example 1.*

monotonous[4]. Notably, they are also completely *density-invariant*, since the influence of an example point can easily extend out further in some directions than in others; it never overlap other example point, and it is always convex in shape. Figure 6.10 shows a set of example points interpolated with Cartesian gradient band interpolation. It can be seen that the weight function of an example point extends out exactly far enough to touch the neighboring example points, without overlapping them, even though those neighboring points are not equally far away.

6.3.1 Gradient Bands in Polar Space

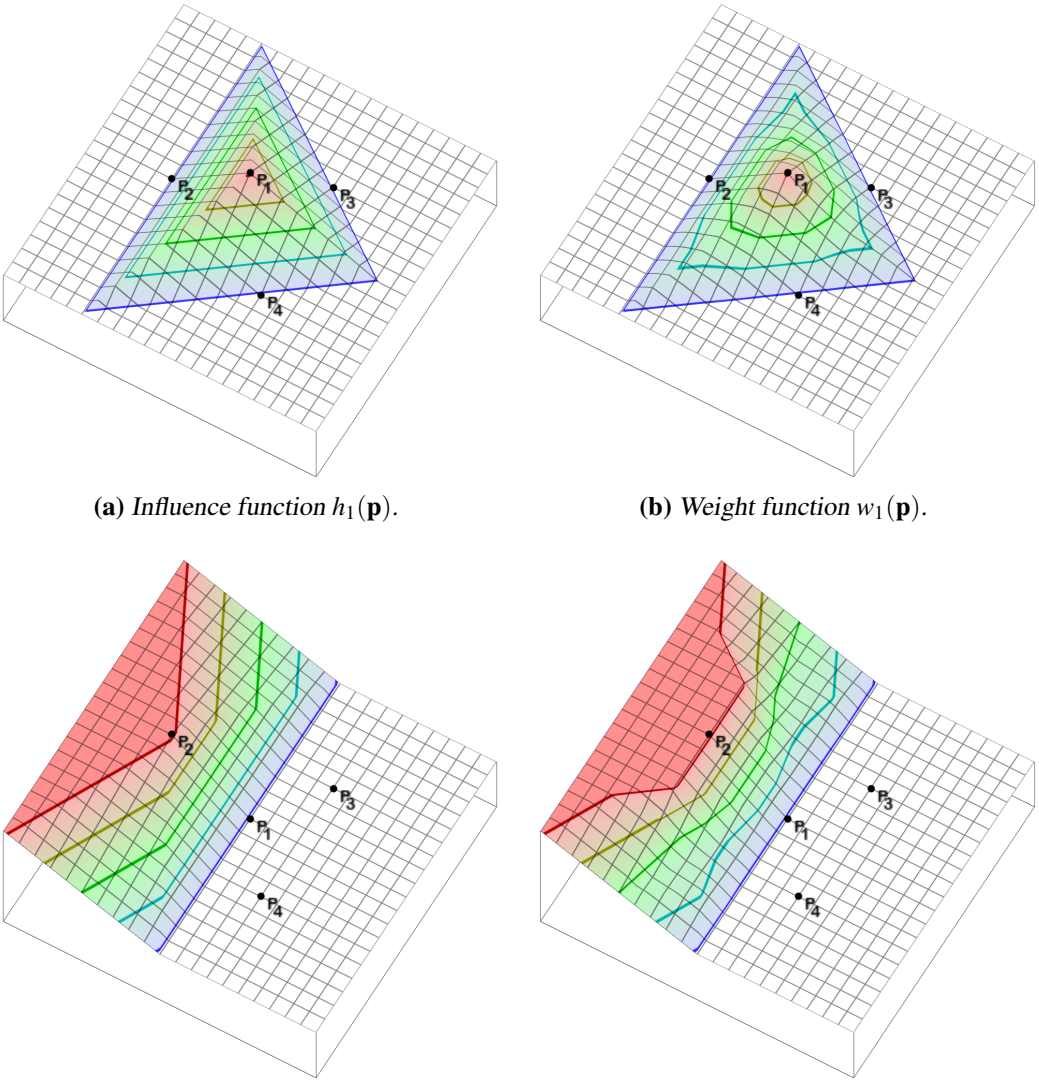
Initially, the influence functions were defined in standard Cartesian space with the vectors $\overrightarrow{\mathbf{p}_j\mathbf{p}_i}$ and $\overrightarrow{\mathbf{p}_j\mathbf{p}}$ specifying the differences in the x, y, and z components of the points:

$$\overrightarrow{\mathbf{p}_i\mathbf{p}_j} = (x_j - x_i, y_j - y_i, z_j - z_i) \tag{6.8}$$

$$\overrightarrow{\mathbf{p}_i\mathbf{p}} = (x - x_i, y - y_i, z - z_i) \tag{6.9}$$

However, the standard gradient band interpolation is not well suited for interpolation of examples based on velocities. As can be seen in figure 6.11, the influence of the weight function $w_4(\mathbf{p})$ of $\mathbf{p}_4$, (which is a walking example motion) extends out to speeds that exceed the running speeds of examples $\mathbf{p}_8$ and $\mathbf{p}_9$ (which are

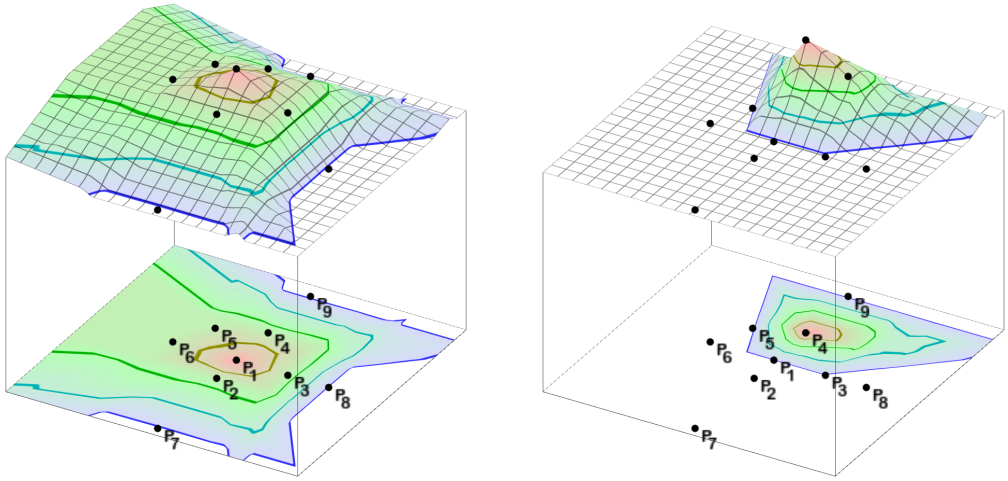
[4]The weight functions have been monotonous in practice, though cases may exist where this is not the case. The influence functions are always monotonous prior to normalization.



(a) *Influence function* $h_1(\mathbf{p})$. **(b)** *Weight function* $w_1(\mathbf{p})$.

(c) *Influence function* $h_2(\mathbf{p})$. **(d)** *Weight function* $w_2(\mathbf{p})$.

Figure 6.9: *Gradient band interpolation with four examples. The influence and weight functions are shown for example 1 and 2. The influence functions are the weight functions prior to normalization. Note that the extends of the weight functions (marked by the outer blue lines) are not altered by the normalization.*



(a) *Interpolation of heights.*

(b) *Weight function $w_4(\mathbf{p})$. The weight function extends out exactly far enough to touch the neighboring example points, without overlapping them, even though those neighboring points are not equally far away from $\mathbf{p}_4$.*

Figure 6.10: *Cartesian gradient band interpolation.*

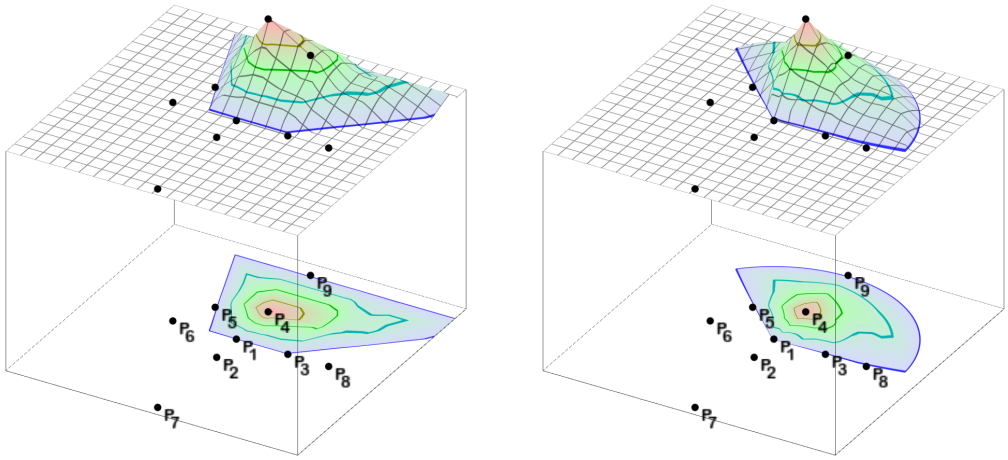
running example motions) by far. The Polar gradient band interpolation proposed here overcomes this problem.

The Polar gradient band interpolation method is based on the reasoning that in order to get a more desirable behavior for the weight functions of example points that represent velocities, the space in which the interpolation takes place should take on some of the properties of a polar coordinate system. This means that this interpolation method is far less general in that it is based on a defined origin in the interpolation space. However, in the specific case of handling velocities, this can be an advantage, since it allows for dealing with differences in direction and magnitude rather than differences in the Cartesian vector coordinate components. In this polar space, the gradient bands behaves somewhat differently:

- Two example points placed equally far away from the origin should have a wedge shaped gradient band between them that extends from the origin and outwards (figure 6.12a). In this case the direction of the new point relative to the two example points determine the respective weights of the two examples.

- Two example points placed successively further away from the origin in the same direction should have a circle shaped gradient band between them with its center in the origin (figure 6.12b). In this case the magnitude of the new point relative to the two example points determine the respective weights of the two examples.

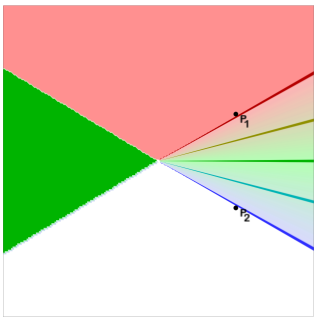
In the same manner that horizontal and vertical lines can be generalized in the form



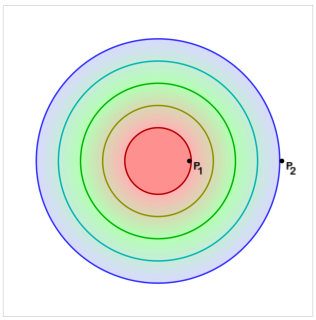
(a) *Weight function $w_4(\mathbf{p})$ for Cartesian gradient band interpolation. The influence of $\mathbf{p}_4$ reaches out to velocities that far exceed the velocity of its neighbor point $\mathbf{p}_9$ that has greater velocity. This means that walking example motions such as $\mathbf{p}_4$ can have an influence at velocities where only running example motions such as $\mathbf{p}_9$ should have any influence.*

(b) *Weight function $w_4(\mathbf{p})$ for Polar gradient band interpolation. The influence of $\mathbf{p}_4$ is nicely confined to velocities that are not any greater than the velocity of its neighbor point $\mathbf{p}_9$.*

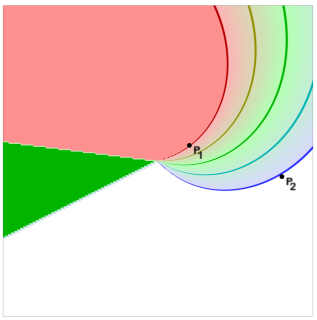
Figure 6.11: *Weight function $w_4(\mathbf{p})$ for Cartesian gradient band interpolation and Polar gradient band interpolation, respectively. The example points represent velocities of example motions, with $\mathbf{p}_1$ placed at the origin with zero velocity. Points further away from the origin have higher velocities.*



(a) *Gradient band extending from the origin and outwards.*



(b) *Gradient band forming a circle around the origin.*



(c) *Gradient band in the form of an Archimeddean spiral.*

Figure 6.12: *Gradient bands in polar space. (Note that while the interpolation is ill-defined in certain areas when there is more than 180 degrees between two adjacent example points, (seen as flat green areas in sub-figure a and c), such ill-defined areas do not show up as long as there are no more than 180 degrees between adjacent example points.)*

of arbitrary lines, circles and lines extending from the origin can be generalized in the form of Archimeddean spirals. An arbitrary line can be described by this parametric formula in 2D Cartesian coordinates:

$$x = x_0 + at \tag{6.10}$$
$$y = y_0 + bt \tag{6.11}$$

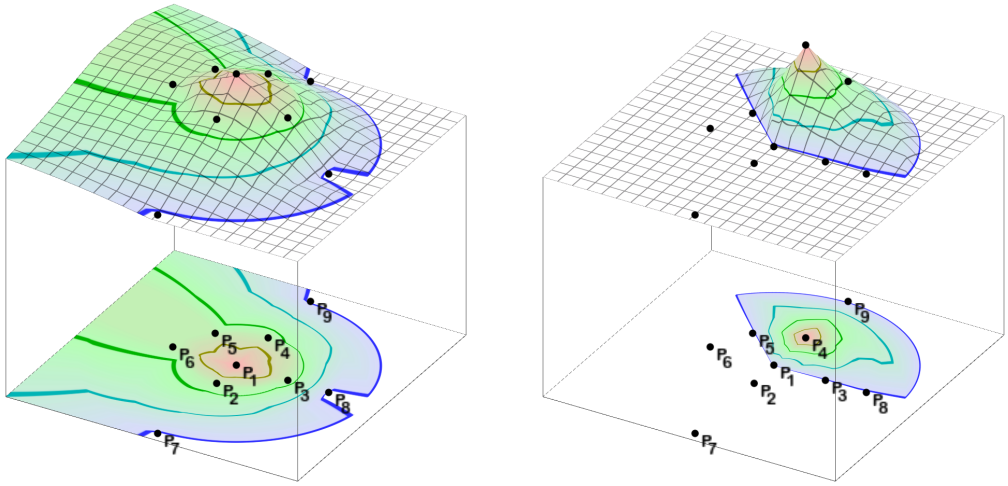
Note that the equation defines a vertical line when $a = 0$ and a horizontal line when $b = 0$. Similarly, an arbitrary Archimeddean spiral can be described by this parametric formula in 2D polar coordinates:

$$r = r_0 + at \tag{6.12}$$
$$\theta = \theta_0 + bt \tag{6.13}$$

This equation defines a circle when $a = 0$ and a line going through the origin when $b = 0$. Thus, to generalize the two properties listed above, any two example points should have a gradient band between them in the form of a (generalized) Archimeddean spiral (see figure 6.12c).

In order to obtain this form of gradient bands, the vectors $\overrightarrow{\mathbf{p}_i\mathbf{p}_j}$ and $\overrightarrow{\mathbf{p}_i\mathbf{p}}$ are redefined from specifying differences in Cartesian coordinates to specifying differences in magnitudes and directions (angles), similar to polar coordinates:

$$\overrightarrow{\mathbf{p}_i\mathbf{p}_j} = \left(\frac{|\mathbf{p}_j| - |\mathbf{p}_i|}{(|\mathbf{p}_j| + |\mathbf{p}_i|)/2}, \angle(\mathbf{p}_j, \mathbf{p}_i)\alpha \right) \tag{6.14}$$
$$\overrightarrow{\mathbf{p}_i\mathbf{p}} = \left(\frac{|\mathbf{p}'| - |\mathbf{p}_i|}{(|\mathbf{p}_j| + |\mathbf{p}_i|)/2}, \angle(\mathbf{p}', \mathbf{p}_i)\alpha \right) \tag{6.15}$$



(a) *Interpolation of heights.* **(b)** *Weight function* $w_4(\mathbf{p})$.

Figure 6.13: *Polar gradient band interpolation.*

where, in the 3D case, $\mathbf{p}'$ is $\mathbf{p}$ projected onto a plane with the normal defined by the cross product $\mathbf{p}_i \times \mathbf{p}_j$ and where both $\angle(\mathbf{p}_j, \mathbf{p}_i)$ and $\angle(\mathbf{p}', \mathbf{p}_i)$ are signed angles in radials around the same axis $\mathbf{p}_i \times \mathbf{p}_j$. In the 2D case $\mathbf{p}' = \mathbf{p}$, and $\angle(\mathbf{p}_j, \mathbf{p}_i)$ and $\angle(\mathbf{p}', \mathbf{p}_i)$ are signed angles in radials in the 2D plane. Note that there are no absolute angles used despite the references made to polar coordinates. Only angles between vectors are considered, and the interpolation works well in both 2D and 3D. The multiplier α controls the relative importance of directions and magnitudes.

Angles are inherently scale independent; two vectors will have the same angle between them whether the vectors are measured in centimeters or meters. In order to achieve the same scale independence for the magnitude differences, such that angles and differences in magnitudes can be meaningfully compared, the difference in magnitudes between two velocities is divided by the average magnitude of the two points, in this case $\mathbf{p}_i$ and $\mathbf{p}_j$, as can be seen in equation 6.14 and 6.15.

The multiplier α controls the relative importance of directions and magnitudes in the interpolation and I found through experimentation that the most intuitively desirable results are achieved with $\alpha = 2$, such that the interpolation is affected twice as much by direction as by magnitude (with angles measured in radians).

The interpolation is ill-defined in certain areas when there is more than 180 degrees between two adjacent example points, either clockwise or counter-clockwise (or any way around in the case of 3D). See figure 6.12. However, as long as there are no more than 180 degrees between adjacent example points, these ill-defined areas do not show up. Figure 6.13 shows a set of example points interpolated with polar gradient band interpolation. As can be seen, the influence of the weight function $w_4(\mathbf{p})$ of $\mathbf{p}_4$, (which is a walking example motion) is confined to the walking range of speeds, while it has little or no weight when the speed exceeds the running speeds of examples $\mathbf{p}_8$ and $\mathbf{p}_9$ (which are running example motions).

6.4 Results

The Gradient Bands in Polar Space interpolation method has been used on a variety of characters with different amounts of example motions (see chapter 8 for details). In all of the sets, the method provides intuitive interpolation of the motions in real time with continuously changing parameters. In all cases the interpolation is smooth as long as the parameter changes smoothly. While the method in its naive implementation that was used for this work performs in $O(n^2)$, where n is the number of example motions, it can be reduced to constant time like all strictly local interpolation methods, given a proper look-up table over which examples have non-zero influence in which areas of the parameter space. This makes it comparable in speed to k-nearest-neighbors interpolation, and densely sampled parameter space, and faster than cardinal radial basis functions. Furthermore, in all the experiments conducted the method has performed well in real-time even in its naive implementation due to the low number of example motions used.

6.5 Summary of Motion Interpolation

For interpolation of example motions with different parameters, scattered data interpolation can be used to find desirable blend weights. In this work, the example motions are walk and run cycles (and possibly an idle motion) and the parameters are the components of a vector specifying the velocity of a motion. Given an arbitrary velocity at runtime, the scattered data interpolation method should provide a set of blend weights that can be used to blend the example motions to obtain a good-looking blended motion.

A list of interpolation constraints has been formulated to describe the desired properties for the interpolation method. Several existing interpolation methods have been reviewed with regard to these constraints and have been found, in different ways, to not live up to all the desired constraints. A new method has been proposed, referred to as Gradient Band Interpolation, and its desirable properties with regard to the list of desired constraints has been demonstrated. A special variant of the method referred to as Gradient Band Interpolation in Polar Space have also been proposed, where distances between different points in the parameter space are measured in terms of differences in direction and magnitude rather than differences in the Cartesian vector coordinate components. This work argues that such an interpolation is particularly suitable for the specific case of interpolating walk and run cycle motions with different velocities.

The Gradient Bands in Polar Space interpolation has been successfully used in a number of different cases with characters with different numbers of example motions (see chapter 8 for details). The method has been found to generally work in an intuitive way regardless of the number of example motions used.

Chapter 7

Semi-Procedural Animation

In the previous chapters, methods have been described for analyzing certain properties of provided example motions, and for blending multiple example motions with proper synchronization and with suitable blend weights. This chapter proposes semi-procedural methods for adjusting the blended motion such that the character can walk or run in any direction on any uneven terrain. Methods are described for predicting where the feet should land on the ground next; for calculating the trajectories and alignments of the feet when taking steps, and for adjusting the legs and hip height to accommodate the new foot positions. All the methods are designed to be faithful to the original motion, meaning that the motion is not adjusted any more than necessary.

7.1 Stepping Properties

Looking again at the leg cycle, the keytimes that were analyzed as part of the motion analysis will come into use at runtime when calculating the semi-procedural motion for the character. The keytimes of the leg cycle were described in chapter 4. At runtime, the progression through various phases in the leg cycle are represented in the these time values (see figure 7.1b):

Leg cycle time: A value increasing linearly from 0 at stance time to 1 at the next stance time.

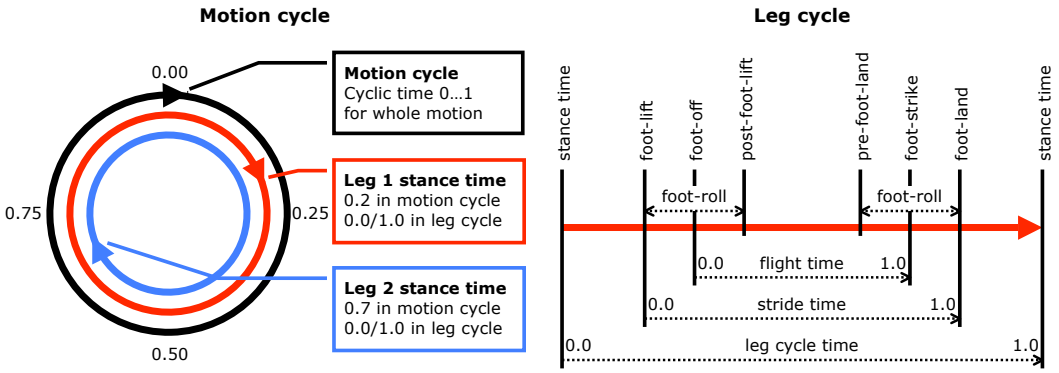
The leg cycle time is defined relative to the motion cycle time, such that leg l is at its stance time when the leg cycle time is 0 or 1:

```
1 leg[l].cycleTime =
2    cyclic(motionCycleTime - leg[l].stanceTime);
```

Listing 7.1: *Cycle time of leg* l.

where $\text{cyclic}(k) = k - \text{floor}(k)$. The following values are subsets of the span of the leg cycle, and are defined relative to it:

Stride time: A value increasing linearly from 0 at *foot-lift* to 1 at *foot-land*.



(a) *Leg cycles relative to motion cycle. This example shows two legs with stance times opposite each other in the motion cycle.*

(b) *Keytimes in leg cycle. Each leg has a leg cycle with the leg cycle time going from 0.0 at the stance time to 1.0 at the stance time again.*

Figure 7.1: *Motion cycle and leg cycles.*

Flight time: A value increasing linearly from 0 at *foot-off* to 1 at *foot-strike*.

Furthermore, the following general concepts are central to the semi-procedural movements of the feet and legs:

Footprint: A place on the ground where the foot steps on at a specific time.

Stride: A single step from one footprint to the next.

7.2 Feet Placements on the Ground

When walking, we look more at the ground in front of us than we think, constantly anticipating approximately where we will step on the ground next. Walking uphill, we lift our feet more to not swing them into the ground, and whenever we step over a door frame even just an inch high, we lift the foot in order to not stumble over it. Walking down stairs is easy, except when carrying a large box that prevents us from looking down, in which case a miscalculation of the number of steps left can cause us to almost fall over if there turns out to be one more step left than we thought. Anticipation is key.

In this respect, taking a step is a matter of lifting the foot from one specific place on the ground (the *footprint*) to another. Making an algorithm out of this, each foot can be said to have two footprints assigned at any time: The *previous* and the *next* footprint. The foot makes a stride from the previous footprint to the next, and when the leg reaches its *stance* time in its leg cycle, its *next* footprint is immediately turned into its *previous*, and a new *next* is assigned.

7.2.1 Related Work

Prediction of the next footprint of a foot has been covered in various forms in previous literature. kai Chung and Hahn (1999) has developed a hierarchical and procedural motion control system to generate walking motion along a path on uneven terrain, continuously planning the footprints two steps ahead to allow anticipation of obstacles ahead (kai Chung and Hahn, 1999, p4). Similarly, Sun and Metaxas (2001) has developed a procedural system for generating human gaits along a specified path on uneven terrain, determining step lengths based on prediction of where the current swing foot will land (Sun and Metaxas, 2001, p267). These two methods are both targeted specifically at human characters and are both mainly procedural, meaning that although personality can be varied somewhat by tweaking parameters in the procedural calculations, skeletal motion cannot be specified directly in the form of motion capture or keyframed animation. (In the case of Sun and Metaxas (2001), motion data can be obtained from a source, but it is represented in a form of sagittal elevation angles, which does not retain all motion details.)

Gleicher (2001b) has discussed the implications of editing the path of a character while preserving the original constraints, such as feet planted on the ground without footskate. As the position and alignment of the character is transformed in each frame to get the character walking along a specific path, the footsteps on the ground (in this work referred to as footprints) can be transformed together with the character. He points out however, that since footprint constraints typically spans over a duration of time rather than just a single instant of time, and since the position of the footprint constraints should not move over that duration in order to avoid footskate, the transformation of the character at a single point in time must be chosen and then applied to the constraint over its entire duration. Gleicher suggests that the three most obvious choices are the *beginning*, *middle*, and *end* of the duration of the constraints (Gleicher, 2001b, p5). (Gleicher also briefly discusses the importance of keeping the constraints of the heel and toe synchronized in order to avoid artifacts. The method in this work circumvents that issue entirely by using the unified *footbase* constraint which controls both heel and toe.)

While Gleicher has proposed methods to use for offline editing of motions, footstep constraints are just as relevant for online interactive character motion. Forsyth (2004) proposes a method of determining footstep constraints that is based on prediction of where the foot will land on the ground next. The foot is then constrained to that constraint in the time span between *foot-strike* and *foot-off*. For straight walking, he proposes using the *foot-strike* time as the reference for the footprint (Forsyth, 2004, p25). This corresponds to using the *beginning* of the duration of the footprint constraint as the time of reference to base the position and alignment of the footprint constraint on. However, for curves walking and strafing, Forsyth proposes using the mid-step instead (the *middle* of the duration). This is so that the foot is pointed in the direction of the body and underneath the center of gravity midway through a step (Forsyth, 2004, p26). It is unclear from the source why the mid-step is not simply used in all cases, including straight walking.

7.2.2 Footprint Prediction

In this work, the *stance time* is chosen as the time of reference to base the position and alignment of the footprint constraint on. The stance time may occur at any time between the beginning and end of the stance duration, but is usually somewhere around the middle, as determined by the stance time heuristic (see sub-section 4.3.2 on page 31). This means that the pose of the character will be closest aligned with the corresponding pose in the original motion at the stance time. This is intuitively desirable, as the balance of the character with respect to a given leg is most important to be kept true to the original motion when that leg is standing most firmly on the ground, and less important when the foot has only just stroked the ground, or when it is about to lift off the ground.

When a character changes velocity or rotational velocity suddenly during a step, the prediction of the next footprint should change accordingly. Thus, the time, position, and orientation of the next footstep is continuously predicted and updated right up to the *foot-strike* point in time when the foot touches the ground and has finished its flight phase. The position and orientation of the previous footprint, on the other hand, are static and do not change once they have been assigned.

The predicted time of the *next* footprint of leg *l*, `nextFootprintTime`, is found by multiplying the cycle duration with the remaining leg cycle time until the stance time is reached, and adding that to the current time:

```
1 leg[l].nextFootprintTime =
2    time + cycleDuration * (1 - leg[l].cycleTime);
```

Listing 7.2: *Time of next footprint for leg* l.

Given the time of the next footprint, the predicted position and orientation of the character in world space at that time can be found:

```
1 leg[l].nextFootprintCharacterPosition =
2    characterPosition
3    + (characterVelocity * (leg[l].nextFootprintTime - time));
4
5 leg[l].nextFootprintCharacterRotation =
6    (characterRotationalVelocity * (leg[l].nextFootprintTime - time))
7    * characterRotation;
```

Listing 7.3: *Predicted position and rotation of character at time of next footprint for leg* l.

For simplicity, the above code listing predicts the future position of the character based only on the velocity, while the rotational velocity is not used. This gives reasonable results in many cases. In later versions of the implemented system, the rotational velocity is taken into account in the prediction too by calculating the radius of a circular path that the character is predicted to be moving along. The code for this is left as an exercise for the reader.

Given the predicted position and orientation of the character, the predicted position and orientation of the footprint in world space can be found as well:

```
1 leg[l].nextFootprintPosition =
```

```
    leg[l].nextFootprintCharacterPosition
    + leg[l].nextFootprintCharacterRotation * leg[l].stancePosition;

leg[l].nextFootprintRotation =
    leg[l].nextFootprintCharacterRotation * leg[l].stanceRotation;
```

Listing 7.4: *Predicted position and rotation of next footprint for leg* l.

Since the predicted positions and orientations can change rapidly, smoothing can be necessary to ensure smooth continuous changes. The influence of the predicted footprint has increasing influence on the actual position of the foot as the current time gets nearer to the predicted footprint time, so in the implemented system the smoothing is increased accordingly.

7.2.3 Grounded Footprints

The predicted position `nextFootprintPosition` and orientation `nextFootprint-Rotation` of the footprint, as described above, are based only on the current position, velocity, orientation, and rotational velocity of the character. To make the footprint be positioned on the surface of the ground, the collision system of the game engine can be used to find the height and normal vector of the ground surface at the predicted position. In the implemented system, raycasts[1] are used for this purpose. The raycasts check for intersections with the ground along a ray starting a specified distance above the predicted position and pointing directly downwards. A grounded predicted footprint position and orientation is then calculated such that the footprint is placed on the ground and aligned with the surface of the ground. In the following, the same variables `nextFootprintPosition` and `nextFootprintRotation` will refer to the grounded position and orientation, since after the grounding there is no longer any use of the original non-grounded position and rotation.

As a compromise between efficiency and accuracy, two raycasts are used for each foot in the implemented system, corresponding to the heel and toe positions. Some basic checks are performed to make sure that the spots hit for both the heel and toe have valid degrees of steepness and that any ground is hit there at all.

7.3 Footbase Trajectories

When taking a step, the trajectory of the foot from the previous footprint to the next is dependent on many factors. While in contact with the ground, the foot should be correctly constrained by the current footprint position and alignment. This can be handled with the use of inverse kinematics. However, when lifting off a footprint, or landing on the next, the motion must be continuous and not snap into the constrained

[1] A raycast is a feature present in many modern game engines which determines the first object intersected by a ray. Given the starting point, direction, and possibly a length of this ray, the first intersection is returned with information about its position and the normal of the intersected surface at that position. Specific implementations may offer additional options and information.

pose, but rather animate gracefully into it. Other challenges include the issues of making the foot trajectory look natural in accordance with the overall character trajectory, with the uneven terrain, and with the trajectories of the other feet, while still being true to the original keyframed animation.

7.3.1 Related Work

The problem of moving the legs of an animated character according to an original motion but constrained such that the feet must correctly land and stay on the ground without footskate has been the subject of much prior work.

An optimization-based scheme was proposed by van de Panne (1997) to generate biped locomotion from given footprints. With the scheme, the motion is represented solely in terms of a center of mass trajectory which itself is synthesized from the footprint information. This scheme was further extended to quadruped locomotion (Torkos and van de Panne, 1998). Compared to the method described here, these methods differ in that the overall character trajectory (center of mas) is synthesized from the footprint positions, while in this work, the footprint positions are generated and predicted based on the character trajectory.

kai Chung and Hahn (1999) has presented a fully procedural method for human locomotion. Foot trajectories from one footprint to another with minimal energy spent are calculated based on bezier curves with control points raised just enough to clear the uneven terrain. The height of the hips was based on the current support leg. The method is not based on example motions.

Gleicher (2001a) has compared different constraint-based motion editing techniques. The broad category of *per-frame inverse kinematics* is any technique that applies inverse kinematics to a configuration in each frame, rather than only at certain key frames with interpolation applied in between. Since an interactive medium such as computer games create situations where the exact constraints cannot be known in advance, it is necessary to enforce that the constraints are satisfied in each frame.

Gleicher (2001a) has found a category of techniques called *per-frame inverse kinematics plus filtering* (PFIK+F), first presented by Lee and Shin (1999), to have many benefits. PFIK+F methods look individually at each frame and solves the IK problem in that frame such that all current constraints are met. After this, a global process considers frames close to each other in time and seeks to remove the temporal artifacts introduced in the first step, typically by performing a low-pass filter meant to remove spikes and other discontinuities introduced in the curves by the independence of the adjustments in the first step (Gleicher, 2001a, p114). As suggested by Gleicher (1998, p5), Lee and Shin apply the filter not to the motion as a whole, but rather to the deltas in motion introduced by the constraints, such that high-frequency features of the original motion are kept intact to the extent it is possible within the constraints. Since the per-frame IK solver and the temporal filter destroy the work of each other, the process is iterated a number of times.

PFIK+F is a very general method that can be used for a wide range of pur-

poses and motions. However, the quality of the results are highly dependent on the specifics of the parameters that the filtering pass is applied with. For example, the amount of filtering applied in the various iterations of the solver can make the difference between a motion of good quality and one where a character shows noticeable "lunging" to meet desired footprints (Gleicher, 2001a, p123). Furthermore, PFIK+F is inherently offline (Gleicher, 2001a, p124).

For moving a foot from the current/last footprint to the next in a realtime simulation, Forsyth (2004) proposes a method which is also based on PFIK but without filtering in the traditional sense. The position and orientation of the predicted next footprint constraint is compared to where the foot would land without this constraint, and the delta of the heel position and foot orientation is calculated in character space. The same is done for the known current/last footprint constraint. Rather than applying filtering to the deltas, they are applied with a multiplier "strength" between zero to one. *A priori* knowledge of the cycle of the stride is used to determine which deltas are applied at any given time, and with which strength. While the foot is planted, the delta required to meet the current(/last) footprint constraint is applied with full strength. While the foot is in the first half of its swing, the delta required to meet the (current/)last footprint constraint decreases from full to zero strength, and in the second half of the swing the delta required to meet the next footprint constraint increases from zero to full strength (Forsyth, 2004, p28-32).

7.3.2 Overview

Given the positions, orientations, and points in time of the previous footprint as well as the next predicted footprint, the footbase needs to move from the previous to the next footprint during the course of the *flight* of the foot from *foot-off* to *foot-strike*. This must be done in accordance with the blended animation, yet also in accordance with the positions and orientations of the given footprints in world space. This is where the normalized footbase trajectories found in the analysis phase comes into play, as described in the following sub-section. Applying the normalized footbase trajectories while also taking the possibly uneven terrain into account is a complex problem that I have divided into multiple sub-problems which will be discussed in the following sub-sections.

1. Applying the original flight progression.
   Use the progression from the normalized footbase trajectory to keep the same acceleration and deceleration of the footbase as in the original motion.

2. Character following trajectory curves.
   Add an offset to the straight line footbase trajectory to make it follow the curve the character is walking in.

3. Ground tangent based lifting.
   Lift the footbase in a suitable arc such that it does not intersect with the (uneven) ground.

4. Lifting to height of supporting feet.
   If the supporting leg or legs are standing on a higher elevated surface than the lifting foot, then lift the lifting foot in an arc up to that same height.

5. Applying the original lifting and sideways movements.
   On top of the trajectory calculated so far, add the sideways and lifting movement from the normalized footbase trajectory in order to stay faithful to the original motion.

7.3.3 Applying the Original Flight Progression

Starting with the most naive implementation of the horizontal footbase flight path, I will in the following discuss various approaches to the problem. The simplest approach is a linear interpolation between the previous and next footprint, using the *flight time* as the interpolation variable:

```
leg[l].footbasePosition =
Lerp (
   leg[l].prevFootprintPosition,
   leg[l].nextFootprintPosition,
   leg[l].flightTime
);
```

Listing 7.5: *Linear interpolation between previous and next footprint based on flight time.*

with the Lerp function being a function for linear interpolation, defined as:

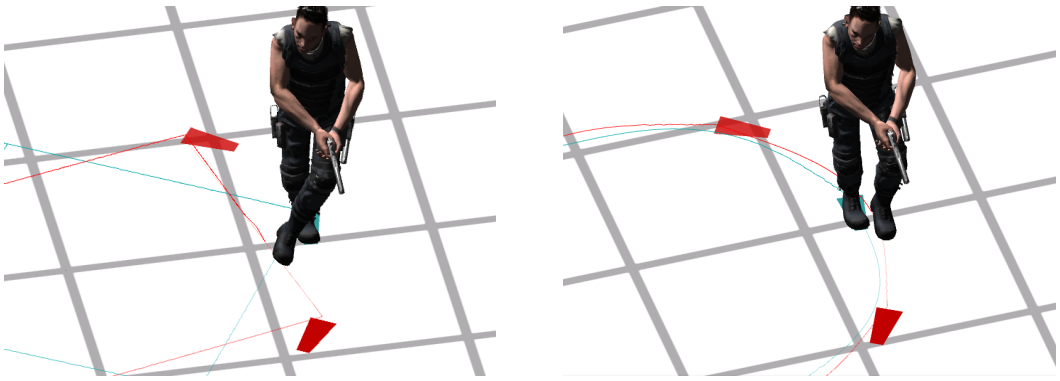
$$\text{Lerp}(a,b,t) = a \cdot (1-t) + b \cdot t \tag{7.1}$$

This, however, is not faithful to the movement curve of the foot in the original motion and thus does not preserve the original characteristics such as the specific acceleration and deceleration of the foot during the flight.

In the chapter on Motion Analysis, it was described how a normalized footbase trajectory is stored for each foot, for each example motion (see sub-section 4.4.1 on page 37). Since the trajectories are normalized, they can be meaningfully averaged, and a weighted average is used with the same weights that are used for blending the example motions at the current moment. To avoid artifacts, the different footbase trajectories are aligned using time-warping, as described in section 5.2 on page 43.

The normalized footbase trajectories sets off at $(0,0,0)$ and lands at $(0,0,1)$. Since the movement of the normalized path is always along the z-axis, the acceleration and deceleration of the foot along its movement axis can be directly derived from the z-component of the position on the normalized footbase trajectory. Using this as the interpolation value for the linear interpolation between the previous and next footprint is already a marked improvement:

```
leg[l].footbasePosition =
Lerp (
   leg[l].prevFootprintPosition,
   leg[l].nextFootprintPosition,
```



(a) *Foot flight trajectory using only linear interpolation between previous and next footprint. The lifted outer leg intersects with the resting inner leg.*

(b) *Foot flight trajectory using additional offset to make trajectory go through the natural mid-flight point for that foot. The lifted outer leg stays clear of the resting inner leg.*

Figure 7.2: *Foot flight trajectory without and with offset.*

```
5     leg[l].normalizedFootbaseTrajectory(leg[l].cycleTime).z
6 );
```

Listing 7.6: *Linear interpolation between previous and next footprint based on the z component of the normalized footbase trajectory.*

7.3.4 Character Following Trajectory Curves

The problem remaining with this approach is that although the speed is not linear, the feet are still moving in a straight line between the previous and the next footprint position. This is no problem when moving straight in one direction, but when turning in an arc it produces unnatural results. When a character (at least a human) walks in an arc, the feet tend to follow that arc round. This is especially important for the outer foot, since if it simply moves in a straight line, it is likely to intersect with the other leg (see figure 7.2a).

In order for the foot to move from the previous to the next footprint in a proper curve, it is desirable that it passes through a point midway through its flight that corresponds to where it would be if it was controlled by the blended motion alone, in other words, its natural mid-flight position (see figure 7.2b). Since the adjustments of each leg is handled separately without regard to the other legs, intersections of the legs cannot directly be prevented, but as long as the blended motion prior to adjustments is intersection free, the motion after adjustments is also very likely to be free of intersections when the foot passes through its natural mid-flight position.

There are many ways to implement a flight trajectory for the foot that passes through its natural mid-flight position. One approach I initially experimented with is to let the original blended motion gradually take over control of the foot towards the middle of the stride. However, in some cases, where the character is walking with a

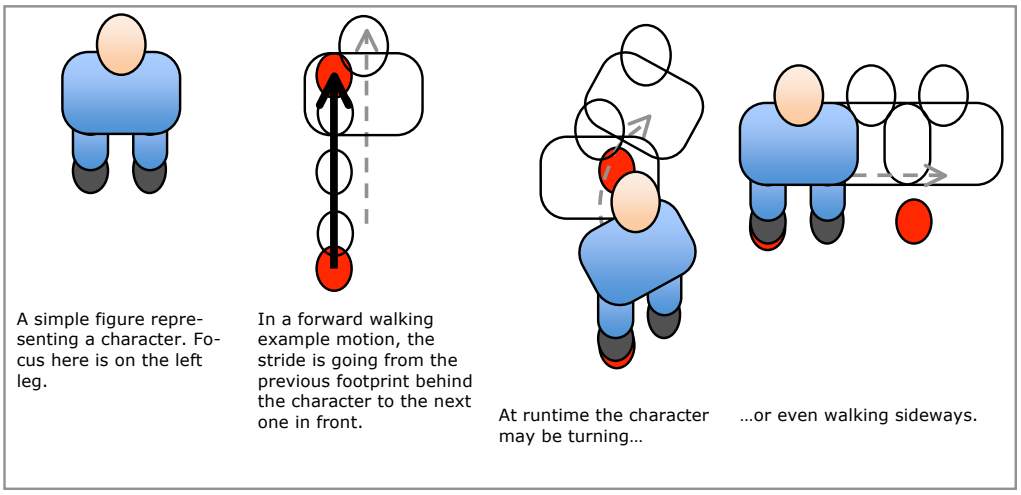


Figure 7.3: *Adjustment of footbase flight trajectory must produce good results no matter if the character is walking straight ahead as in the original motion, or turning, or walking sideways, or walking with any other direction or turning angle.*

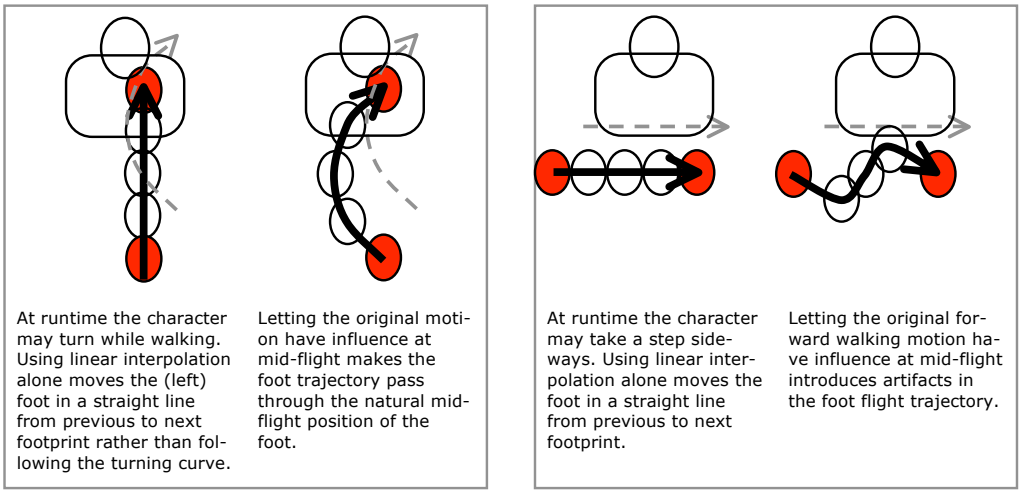
very different speed or direction than in the original motion, this solution has visible flaws. If for example a character is walking sideways based on a forward walking motion, then the forward headed direction of the original motion will affect the trajectory around the middle (see figure 7.4a), which produces unnatural artifacts.

I have eventually come up with a different approach that does not suffer from the flaws of the initial approach described above: A special offset is added to the footbase trajectory, again with maximum weight around mid-flight, that causes it to go through the natural mid-flight position of the foot (see figure 7.4b). The offset vector is the current *stance position* in world space relative to a linear interpolation of the previous and next footprint based on the *leg cycle* time. The interpolated position moves with linear speed from the previous to the next footprint position over the duration of the entire *leg cycle* from *stance time* to the next *stance time*, not just during the stride part of it. This is analogous to how the *character position* moves, at least when moving with constant velocity. That means that as long as the character moves with constant velocity, the offset is always of zero length, no matter if the velocity of the character is the same as the velocity of the original motion or not.

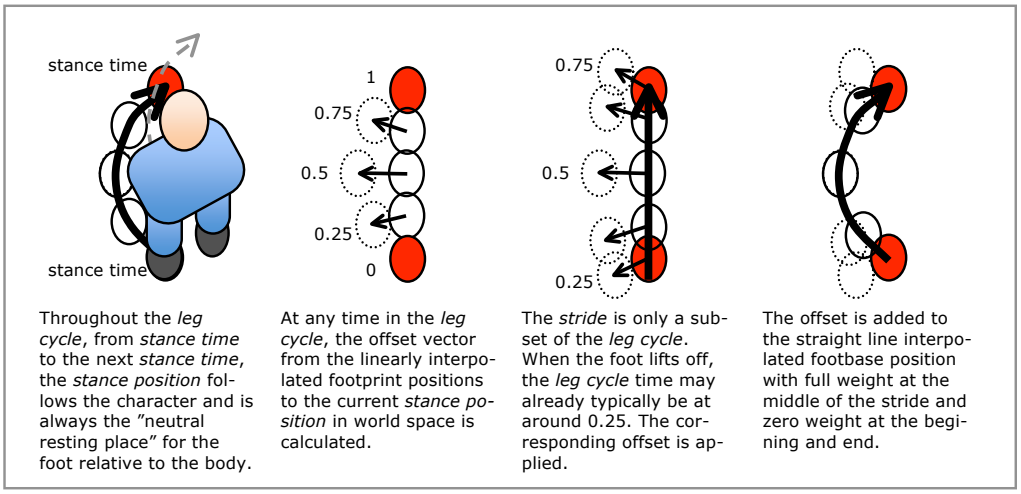
When a character is walking with non-constant velocity, such as accelerating or turning, a smooth weight function for the offset can be used to ensure smooth trajectories. Over the duration of the stride, the offset is added with a weight that is at its maximum midway through the stride, given by the weight function

$$\texttt{flightProgressionArc} = \sin(\texttt{leg[l].normalizedFlightPosition.z} \cdot \pi) \tag{7.2}$$

Using this offset weight function produces very smooth and natural-looking results where the feet trajectories follow the overall path of the character well, as seen in



(a) *Foot flight trajectory using influence of the original motion at mid-flight.*



(b) *Foot flight trajectory using offset vectors at mid-flight.*

Figure 7.4: *Adjustment of footbase flight trajectory using (a) influence of original motion at mid-flight, and (b) offset vectors added at mid-flight.*

figure 7.2b. Adding to the code listings above, the offset is then calculated and added to the footbase position:

```
stancePositionInWorld =
   characterPosition + (characterRotation * leg[l].stancePosition);

interpolatedFootprintPosition =
Lerp (
   leg[l].prevFootprintPosition,
   leg[l].nextFootprintPosition,
   leg[l].cycleTime
);

offset = stancePositionInWorld - interpolatedFootprintPosition;

leg[l].footbasePosition += offset * flightProgressionArc;
```

Listing 7.7: *Interpolation between reference footbase in character space and interpolated footbase in global space.*

7.3.5 Ground Tangent Based Lifting

Since it cannot be assumed that the character is walking on a flat horizontal plane, the terrain under the feet needs to be taken into account. Like in the method described by kai Chung and Hahn (1999), the goal here is to find a foot trajectory curve that clears the terrain with minimal energy spent. Since an exhaustive analysis of the local terrain is deemed too computationally expensive at runtime, a simple approach is taken where only the positions of the footprints are taken into account, as well as the normal of the surface at those positions. This information is available in the `prevFootprintPosition`, `prevFootprintRotation`, `nextFootprintPosition` and `nextFootprintRotation`.

The mid-point between the previous and next footprint is found, and for each of the two footprints the intersections of the plane below the footprint and an axis going vertically up from the mid-point is found (see figure 7.5a and 7.5b). The distance from the mid-point to the highest of those two intersections is used to determine the magnitude of a sinus curve that that lifts the foot up, such that the curve follows the higher of the two tangents (see figure 7.5c). The calculations are as follows, using the previous footprint as example (refer to figure 7.5a and 7.5b):

$$\text{mid} = \frac{\text{prev} + \text{next}}{2} \tag{7.3}$$

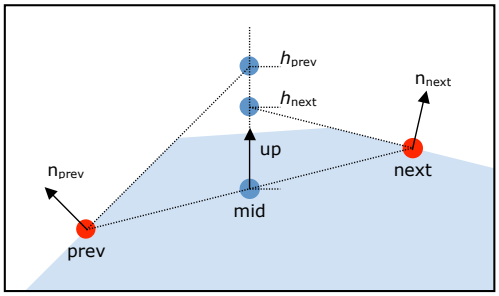
$$\mathbf{a} = \text{prev} - \text{mid} \tag{7.4}$$

$$b = \mathbf{a} \cdot \mathbf{n}_{prev} \tag{7.5}$$

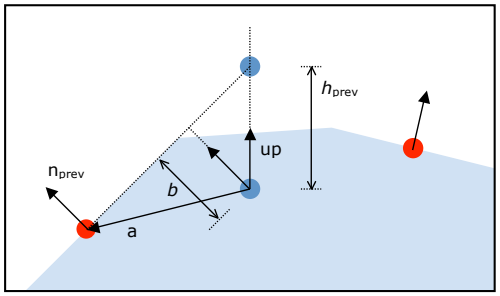
$$h_{prev} = \frac{b}{\mathbf{n}_{prev} \cdot \text{up}} \tag{7.6}$$

where *up* is a global upwards pointing unit length vector[2]. With the heights of the

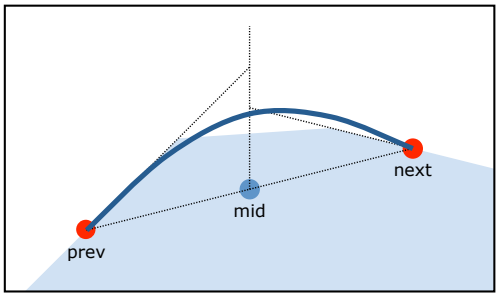
[2]More specifically, the *up* vector should be upwards pointing with regard to the legs. If banking



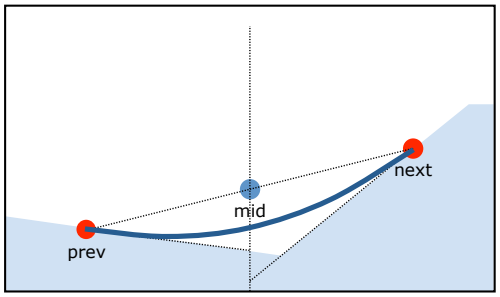
(a) *Intersection of tangents with vertical axis extended from mid-point.*



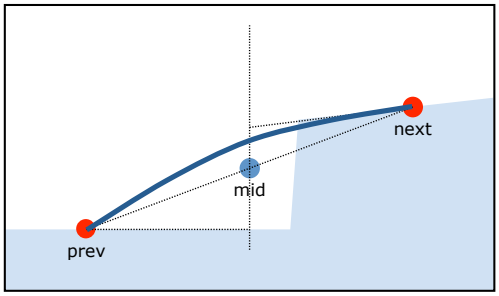
(b) *Calculation of height of tangent intersection above mid-point.*



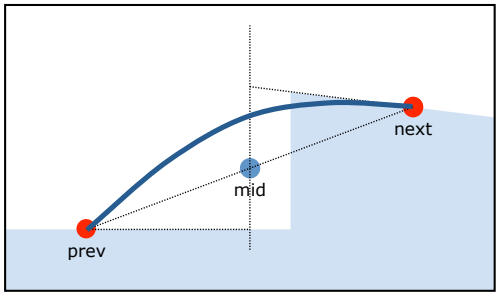
(c) *The sinus curve follows the highest tangent. Its magnitude is based on the height of the higher intersection.*



(d) *Example with valley.*



(e) *Example with a step.*



(f) *Another example with a step; this one requiring higher lifting.*

Figure 7.5: *Lifting of the footbase needed to not intersect with the (uneven) ground.*

Figure 7.6: *The elevation of the body should be primarily dependent on the height of the foot or feet currently grounded. This is achieved by lifting up the foot or feet in mid-flight to the same height as the grounded foot or feet, if they are higher. The height of the body is then later calculated based on the heights of all the feet.*

intersection points calculated, the magnitude of the sinus curve can be found:

$$h_{max} = \max(h_{prev}, h_{next}) \tag{7.7}$$

$$\texttt{arcMagnitude} = \frac{2 \cdot h_{max}}{\pi} \tag{7.8}$$

Adding to the code listings above, the lifting arc is added to the horizontal trajectory of the footbase position defined previously:

```
1 leg[l].footbasePosition += up * arcMagnitude * flightProgressionArc;
```

Listing 7.8: *Lifting arc added to footbase trajectory.*

7.3.6 Supporting Ground Height

The original motion has the torso of the character "hovering" a certain distance above a flat ground. To stay faithful to the original motion, this hover height should be maintained in the adjusted motion at runtime. However, when the character is walking over uneven terrain at runtime, the different feet may step on surfaces of different elevation, and it must be resolved which of the surfaces the elevation of the torso of the character should be measured relative to.

At any point in time while walking, the weight of a character is supported more by some of the legs than others. (More by one leg than the other for a biped.) It is assumed here that all characters have certain configurations for the legs which makes it easiest to carry the weight; for example the supporting leg for a human is

or other features are implemented that tilt the entire character, the up vector might need to match this tilting as well for optimal results. This has been done in the implemented system.

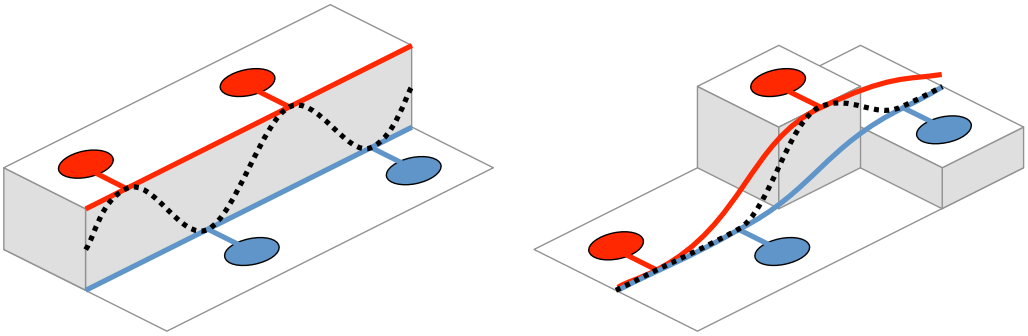


Figure 7.7: *For each foot a smooth* foot ground height trajectory *is created that passes through the footprints for that foot, offset by the negative stance position. A* supporting ground height trajectory *is then calculated as a weighted average of those curves, with full weight given to currently grounded feet, and minimal weight given to feet in mid-flight.*

almost fully stretched while it would require more effort to carry the weight with more bended legs. The legs in mid-swing, which are not currently carrying the weight of the character, can be bended more or less than in the configuration from the original motion with much less effort. For this reason a method is proposed here which seeks to primarily maintain the amount of bending or stretching of the leg or legs that are currently supporting the weight of the character.

The calculation of the actual hover height of the torso of the character is discussed in section 7.4.2 on page 84 and is directly dependent on the positions and alignments of the feet without taking into account which feet are currently supporting the weight of the character. This is instead taken into account in the calculation of the footbase trajectories.

Proposed here is the calculation of a *supporting ground height trajectory* which represent a weighted average of the heights of the surfaces that are currently supporting the weight of the character. Each foot should in mid-swing be lifted up at least to the height of this *supporting ground height*, such that the foot will not pull down the character in the calculation of the torso hover height.

Before the *supporting ground height trajectory* can be found, a *foot ground height trajectory* is defined for each foot l, with respect to the footprints of that foot. The foot ground height trajectory for the foot is not directly related to how the foot itself actually moves but is a smooth curve that goes through the successive footprints of the foot, offset by the negative stance position. At any point in time, a segment of the foot ground height trajectory is defined between the previous footprint and the next.

The foot ground height trajectory segment starts along a tangent to the ground in the previous footprint position and ends along a similar tangent in the position of the next footprint. The directions of the tangents are calculated based on the velocity of the character and on the character rotation at the previous and next footprint relative to the current rotation. The lengths of the tangent vectors are determined by the current cycle distance which equals the length of one step:

```
velocityInCharacterSpace =
   inverse(characterRotation) * velocity;
directionInCharacterSpace =
   (projectedOntoGround(velocityInCharacterSpace)).normalized;

prevFootprintTangent =
   leg[l].prevFootprintCharacterRotation
   * directionInCharacterSpace
   * cycleDistance;

nextFootprintTangent =
   leg[l].nextFootprintCharacterRotation
   * directionInCharacterSpace
   * cycleDistance;
```

Listing 7.9: *Tangents along the ground at previous and next footprint for foot* l.

The smoothness of the curve is obtained by using an S-shaped curve to control the weighting of the two tangents along the length of the curve segment:

```
// S-shaped curve that starts at 0 and ends at 1:
sCurve = -cos(pi * leg[l].strideTime)/2 + 0.5;

leg[l].groundHeightPoint =
Lerp (
   leg[l].prevFootprintPosition
      + prevFootprintTangent * leg[l].cycleTime
   ,
   leg[l].nextFootprintPosition
      + nextFootprintTangent * (leg[l].cycleTime - 1)
   ,
   sCurve
);
```

Listing 7.10: *Calculation of the foot ground height point for foot* l.

In order to get a trajectory with respect to the body origin rather than the foot origin, the curve is offset by the negative stance position:

```
leg[l].groundHeightPoint +=
   characterRotation * -leg[l].stancePosition;
```

Listing 7.11: *Shifting of the foot ground height trajectory by the negative stance position.*

Once the foot ground height trajectory for each foot is calculated, the *supporting ground height trajectory* can be found. The supporting ground height trajectory is a weighted average of the foot ground height trajectories of the individual feet. To simulate robust support on the currently grounded foot or feet, the supporting ground height trajectory is mostly dependent on the foot or feet that are grounded at any given time, while the foot or feet in mid-swing are mostly ignored. The influence h_l of each foot l is defined in relation to the leg cycle time:

$$h_l = \frac{cos(2\pi \cdot \texttt{leg[l].cycleTime}) + 1}{2} + \varepsilon \tag{7.9}$$

Figure 7.8: *Influence of foot* l *on the supporting ground height trajectory as a function of the leg cycle time.*

where ε is a small positive value. This influence for foot l is $1+\varepsilon$ at the stance time, decreases to ε in mid-swing, and increases back to $1+\varepsilon$ at the stance time (see figure 7.8).

The supporting ground height trajectory is calculated as:

$$\texttt{supportingGroundHeightPoint} = \frac{\sum_{l=1}^{legs} h_l \cdot \texttt{leg[l].groundHeightPoint}}{\sum_{l=1}^{legs} h_l} \tag{7.10}$$

For each foot a lifted height is calculated. The trajectory of the lifted height interpolates into the height of the *supportingGroundHeightPoint* halfway through its flight based on a flight time based arc:

$$\texttt{flightTimeArc} = \sin(\texttt{leg[l].flightTime} \cdot \pi) \tag{7.11}$$

Since the lifted height of the *supportingGroundHeightPoint* may be lower than the footbase is lifted to begin with, the footbase is only lifted to the lifted height when the lifted height is higher than the current height of the footbase.

```
liftedHeight =
Lerp (
   leg[l].footbasePosition.height,
   supportingGroundHeightPoint.height,
   flightTimeArc
);

if (liftedHeight > leg[l].footbasePosition.height) {
   leg[l].footbasePosition.height = liftedHeight;
}
```

Listing 7.12: *Supporting feet based lifting added to footbase trajectory.*

7.3.7 Applying the Original Lifting and Sideways Movements

The lifting and sideways movement dictated by the footbase trajectory from the original motion is trivially added in the end after the other factors are taken care of. When moving on a flat horizontal plane the other factors all result in zero offset.

The lifting and sideways movement is then solely based on the normalized footbase trajectory and thus identical to the footbase trajectory in the original motion.

The the sideways movement and the lifting of the footbase from the original motion is added to the footbase trajectory based on the sideways (x) and vertical (y) component of the normalized footbase trajectory analyzed in the motion analysis:

```
1 stepDirection = (leg[l].nextFootprintPosition - leg[l].prevFootprintPosition).
      normalized;
2 stepSidewaysDirection = CrossProduct(up, stepDirection);
3
4 leg[l].footbasePosition += stepSidewaysDirection * leg[l].
      normalizedFlightPosition.x;
5 leg[l].footbasePosition += up * leg[l].normalizedFlightPosition.y;
```

Listing 7.13: *Vertical and sideways movement of footbase from original motion added to footbase trajectory.*

7.4 Leg Adjustments

Once the footbase positions and alignments have been found for each foot, the poses of the legs can be determined. This involves finding the optimal hip height, feet alignments, and using inverse kinematics to find the alignments of the bones between the hip and ankle of each leg.

7.4.1 Related work

kai Chung and Hahn (1999) note that results in gait observation have shown that the ankle joint is generally close to its neutral angle at the times of foot-strike and foot-off (kai Chung and Hahn, 1999, p4), and they go on to describe a method for obtaining natural foot alignments for humans when walking on uneven terrain.

Kovar et al. (2002) proposed a method for finding the root bone offset. Like the method proposed here, they determined the root bone position based on the adjusted ankle positions. However, their method moves the root bone only enough to ensure that the ankle positions can only just be reached by fully extending the legs, and even then it sometimes require a small amount of stretching of the bones. In contrast, the method proposed here determines the position of the root bone such that the bending of the legs in the original motion is maintained as closely as possible, thus staying faithful to the style and personality in the original motion.

The method by Kovar et al. (2002) first calculates the ankle position based on the constraints of the foot in the given frame and the surrounding frames and then calculates where the root should be placed so the ankle positions can be reached by extending the legs (Kovar et al., 2002, p99). This is done by projecting the position of the root onto spheres centered around the positions of the ankles with the respective hip offsets subtracted. The spheres have radii corresponding to the maximum leg lengths (Kovar et al., 2002, p100).

Once both the ankle positions and hip positions are known, inverse kinematics can be used to find the alignments of the bones of the leg in between. There are two general classes of solutions to inverse kinematics problems: Analytic and numerical. Numerical solutions are the most general and are able to handle sophisticated constraints with complex articulated figures. However, they require solving systems of nonlinear equations, so the solutions are typically found iteratively and are computationally expensive. Analytic algorithms find closed-form solutions in fixed amounts of time, but they only exist for simple skeletal configurations (Kovar et al., 2002, p98). Fêdor (2003) have compared different numerical inverse kinematics algorithms and compared the respective cost and quality.

7.4.2 Hip Adjustments

As discussed in section 1.3.1 on page 12, most games today still internally represent a character as a single *object*. To control how fast a character is walking, a force or velocity is assigned to the character *object* as a whole by the game logic. Typically in games that simulate physics, the character is prevented from going through walls or falling through the floor by representing him to the physics system as a single physical object such as a vertically aligned cylinder which can collide with and rest on other physically simulated objects in the game.

Since the legs of the character are typically left out completely of the physical simulation, the altitude of the character *object* above the ground often has no direct relation to where the character actually steps on the ground. Particularly when moving over uneven terrain with steps or staircases, the vertical movement of the character is traditionally jerky or in other ways unnatural looking. In order to make the character look properly grounded, a method is proposed here which determines the altitude of the *root bone* of the character depending alone on the positions and alignments of the feet, regardless of the altitude of the character *object* which would otherwise normally control the altitude of the *root bone*. No horizontal adjustments to the *root bone* position are made, and the altitude of the *root bone* is calculated in a way that completely corresponds to how it is animated in the original motion.

At any point in time in the original motion, there is for each leg a certain distance between the hip and the ankle of that leg. In order to stay faithful to that original motion, that distance from hip to ankle should be maintained as much as is possible in the adjusted motion at runtime. However, when the character is walking over uneven terrain at runtime, it may not be possible to maintain the original distance between hip and ankle simultaneously for all the legs.

Since many factors have influence on the adjusted positions of the feet, they may be offset in all three dimensions relative to the reference position of the foot in the original motion. For each leg, two different potential hip heights are calculated. A desired hip height, *hdesired*, that will maintain how much the leg is bended and a maximum hip height, *hmax*, that will cause the leg to be stretched completely, like in the method by Kovar et al. (2002). This is done by finding intersections between spheres and a vertical line. Specifically, intersections are found of a vertical line

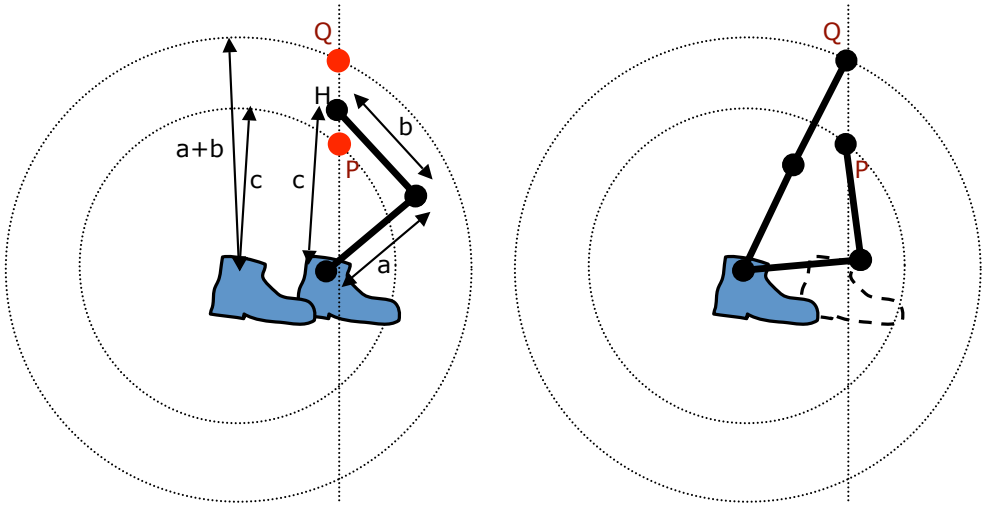


Figure 7.9: *The position of the ankle relative to the hip may be at a different place at runtime than in the original motion and thus the height of the hips may need to be adjusted. For each leg, sphere intersections are used to find the desired hip height P that maintains the original amount of bending of the leg as well as the maximum hip height Q that would make the leg completely stretched. Specifically, intersections are found of a vertical line going through the original hip position and two spheres with their centers at the new ankle position. One sphere has a radius of the original distance between the hip and ankle in the current pose while the other has a radius corresponding to the length of the leg from hip to ankle.*

going through the original hip position and two spheres with their centers at the new ankle position. One sphere has a radius of the original distance between the hip and ankle in the current pose and is used to find *hdesired*, while the other has a the radius corresponding to the length of the leg from hip to ankle and is used to find *hmax* (see figure 7.9). Note that the *hdesired* and *hmax* heights are stored as offsets relative to the hip height in the original motion. Storing the offsets rather than the absolute heights is advantageous since it allows comparing the heights of the different hips, even for characters whose hips do not have the same altitude above the ground; for example animals with longer hind legs than forelegs or similar.

After the desired and maximum hip height have been found for each leg, the following three heights are found: The lowest of the maximum hip heights, the lowest of the desired hip heights, and the average desired hip height.

$$hmax_{min} = \min_{l=1}^{legs}(hmax_l) \tag{7.12}$$

$$hdesired_{min} = \min_{l=1}^{legs}(hdesired_l) \tag{7.13}$$

$$hdesired_{avg} = \frac{\sum_{l=1}^{legs} hdesired_l}{legs} \tag{7.14}$$

The calculation of the final offset of the hips takes these three heights into account based on multiple considerations:

- The naive compromise would be to choose a hip height that is an average of the desired hip heights of all the legs, $hdesired_{avg}$. However, this does not work, since that might cause some of the legs to be hyper-stretched, meaning that the distance between the hip point and ankle point would be longer than the length of the leg from hip to ankle.
- To avoid hyper-stretching, the hip height must be below the lowest of the maximal hip heights of all the legs, $hmax_{min}$.
- The hip height could be chosen as the minimum of $hdesired_{avg}$ and $hmax_{min}$. However, this would produce second-order discontinuities in the hip trajectory whenever the values of $hdesired_{avg}$ and $hmax_{min}$ cross each other, meaning that the motion would not be smooth. Furthermore, whenever $hmax_{min}$ would be chosen as the hip height, at least one leg would be completely stretched, producing stiff and unnatural motion.
- Making a leg be more bended than in the original motion is less likely to produce undesirable artifacts than if it is made to be more stretched. Thus, the lowest of the desired hip heights of all the legs, $hdesired_{min}$, is used as a basis height. It can be assumed that $hdesired_{min}$ is always less than (or equal to) both $hdesired_{avg}$ and $hmax_{min}$.
- The closer the hip height comes to $hmax_{min}$, the more the configuration of one leg will be compromised (the leg that has $hmax_{min}$ as its max height) by making the leg too stiff. At the same time, the further away the hip height is from $hdesired_{avg}$, the less faithful the current adjusted character pose will be from to the current character pose from the original motion.
- The higher $hmax_{min}$ is, the closer the hip height can get to $hdesired_{avg}$ without any legs getting too stiff.

The height differences from $hdesired_{min}$ to both $hdesired_{avg}$ and $hmax_{min}$ are found:

$$mintoavg = hdesired_{avg} - hdesired_{min} \tag{7.15}$$

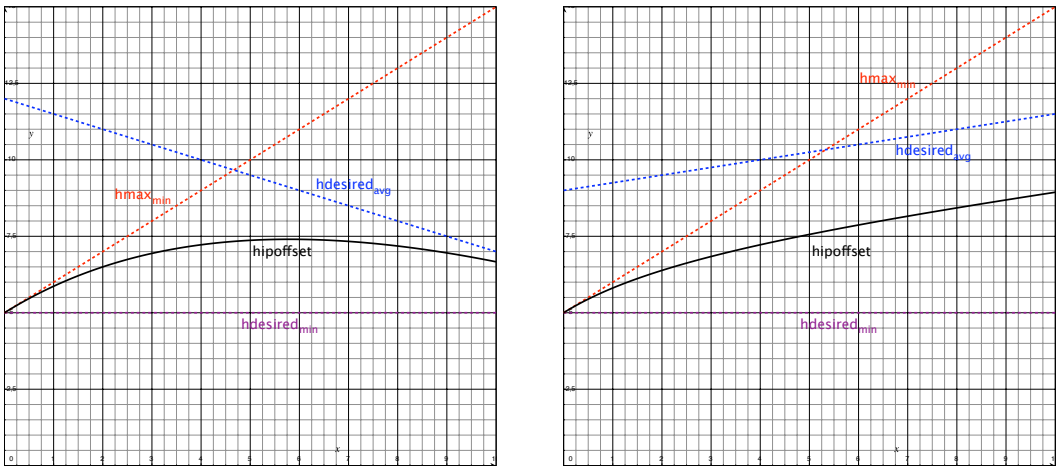
$$mintomax = hmax_{min} - hdesired_{min} \tag{7.16}$$

Then the offset of the hips is calculated by:

$$hipoffset = hdesired_{min} + \frac{mintoavg \cdot mintomax}{mintoavg + mintomax} \tag{7.17}$$

Figure 7.10 shows the *hipoffset* approaching $hdesired_{avg}$ as $hmax_{min}$ approaches infinity and vice versa. The *root bone* of the character is offset with this *hipoffset* along the vertical axis and the hip of each leg inherit this offset.

While the offset of the root bone will often be quite minimal it is not specifically attempted to be minimized, and it can be significant in certain cases. However, as long as the only constraints between the character and the environment are the feet, this is typically not a problem.



(a) *Example 1.* **(b)** *Example 2.*

Figure 7.10: *As the span from* hdesired$_{min}$ *to* hmax$_{min}$ *approaches being many times greater than the span from* hdesired$_{min}$ *to* hdesired$_{avg}$*, the hipoffset approaches* hdesired$_{avg}$*, since when there is no danger of reaching* hmax$_{min}$ *and getting an hyper-stretched leg, the average desired hip height should be closely pursued. Conversely, as the span from* hdesired$_{min}$ *to* hdesired$_{avg}$ *approaches being many times greater than the span from* hdesired$_{min}$ *to* hmax$_{min}$*, the hipoffset approaches* hmax$_{min}$*, since the further the average desired hip height is above, the more stretching of a leg should be tolerated to get closer to that height.*

7.4.3 Foot Alignment

The alignment of the feet of a character throughout a motion is very important to how to overall motion is perceived. Humans tend to roll the feet when walking, hitting the ground with the heel first, then landing the rest of the foot flat on the ground, and finally lifting with the heel first and then the toe. However, certain styles of walking or running may have foot-rolls with different characteristics, and non-human characters may have any odd foot-rolls. Refer to subsection 4.2.1 in the chapter on Motion Analysis for an overview of the generalized keytimes used in this work. Furthermore, I have defined the following phases that describe different time spans in the leg cycle that are useful in the following discussion on foot-roll mechanics:

Stance phase: The phase between *foot-land* and *foot-lift* where the foot is resting flat on the ground and supporting the weight of the character. The *stance time* always occurs at some point inside the time span of this phase, often around the middle of it.

Flight phase: The phase between *post-foot-lift* and *pre-foot-land* where the foot is lifted in its flight from one *footprint* to the next, and furthermore where the foot has finished its foot-roll from the previous footprint and not yet started its foot-roll at the next footprint.

Foot-roll: In theory the time spans from *foot-lift* to *foot-off* and from *foot-strike* to *foot-land*. However, since the the foot may land flat on the ground in the

original motion, with the heel and toe hitting the ground at the same time (or similar for lifting off the ground), thus producing a foot-roll of zero or very short duration, and since such a short foot-roll duration produces bad looking results when modifying the motion, the foot-rolls are enforced to always have a certain minimum length. The keytimes *post-foot-lift* and *pre-foot-land* are defined for this purpose. The lifting foot-roll spans from *foot-lift* to *post-foot-lift* and the landing foot-roll spans from *pre-foot-land* to *foot-land*.

Note that the *stance phase* and *flight phase* do not add up to the entire leg cycle. After the *stance phase* ends and before the *flight phase* begins, there is a transitional *foot-roll* phase, and the same is the case after the *flight phase* ends and before the new *stance phase* begins.

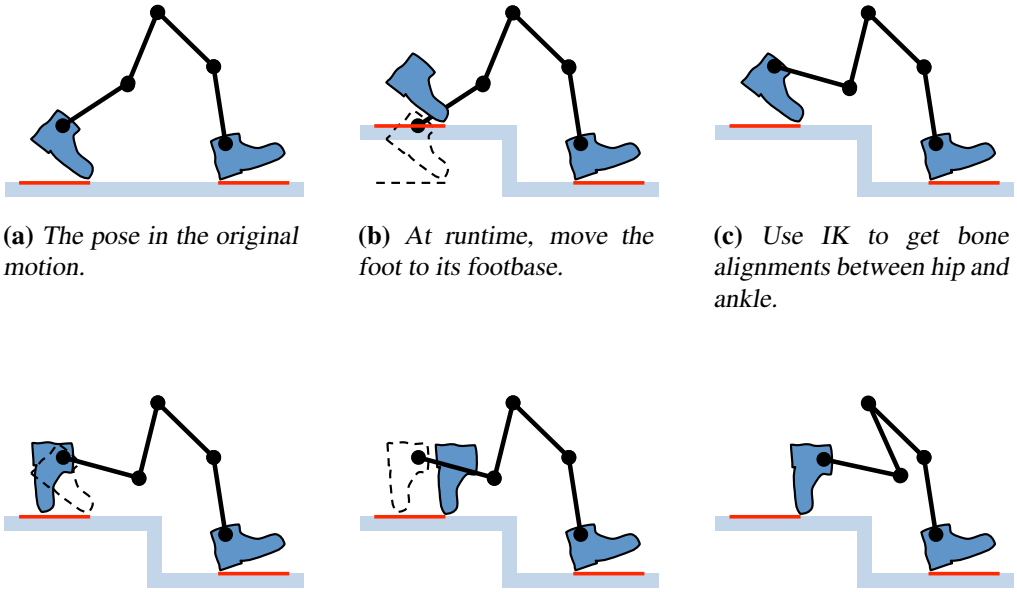
After observing how humans walk over a variety of uneven surfaces, the method proposed here is based on the following propositions:

Ground surface based alignment: In the *stance phase* where the foot is resting flat on the ground in the original motion, it should rest flat on the ground in the adjusted motion too, regardless of the slope of the surface the foot is resting on in the adjusted motion. In other words, the alignment of the foot relative to the ground surface should be kept. This gives an impression of supporting the weight of the character and having maximum friction with the ground.

Ankle joint rotation based alignment: In the *flight phase* where the foot is not touching the ground at all, the adjusted motion should seek to maintain the local rotation of the ankle joint from the original motion. This gives an impression of taking a stride without putting any effort into keeping the foot at any specific angle - the foot is simply in its natural resting alignment relative to the lower leg. Note that this proposition assumes a relatively tight or firm ankle joint (as opposed to loose hanging) that is relatively unaffected by gravity. This assumption seem to hold up fine for the motions of humans and animals that were studied, and also matches the observations made by kai Chung and Hahn (1999, p4).

Foot-rolls: The ground surface based alignment should be used in the *stance phase*. The ankle joint rotation based alignment should be used in the *flight phase*. In the *foot-roll* phases that bridge the *stance phase* and the *flight phase*, an interpolation between the two forms of alignment should be used.

The *ankle joint rotation based alignment* method can be broken down into a few simple steps (see figure 7.11). From the initial alignment of the bones in the leg from the original motion (a), the foot is first moved to its footbase (b). Inverse kinematics are used to find the alignments of the bones between the hip and ankle joints (c). The local rotation of the ankle from the original motion is now applied to the ankle joint (d). The foot is again moved to its footbase (e), and inverse kinematics are again used to find the alignments of the bones between hip and ankle (f).



(a) *The pose in the original motion.*

(b) *At runtime, move the foot to its footbase.*

(c) *Use IK to get bone alignments between hip and ankle.*

(d) *Apply the local rotation from the original motion at the ankle joint.*

(e) *Move the foot to its footbase again.*

(f) *Use IK to get bone alignments between hip and ankle again.*

Figure 7.11: *Steps performed in each frame at runtime to get proper foot-rolls on any uneven terrain while staying as faithful as possible to the original motion. The rotation in (d) is applied with a variable amount that is zero in the stance phase where the foot is standing flat on the ground in the original motion and one when the foot is in its flight phase.*

The *ground surface based alignment* can be implemented using the same steps, except that the rotation in (d) is not applied. Using the same steps for the two methods makes it easy to interpolate between them without extra overhead: The rotation in step (d) is simply applied with a variable amount that is zero during the stance phase, one during the flight phase, and a linearly interpolated value during the foot-roll phases.

The foot alignment obtained with the described method is faithful to the original motion in that the adjusted poses will be identical to the original as long as the ground at runtime matches the reference ground.

7.4.4 Inverse Kinematics

In the above, methods have been described for finding the positions and alignments of the feet and the hips. The alignments of the bones between the hips and ankles are calculated using inverse kinematics.

The details of the IK method is not a primary focus in this work, and the IK system is simply treated like a black box that takes hip and ankle positions, as well as the original configuration of joint angles (the pose) and as outpout produces the adjusted configuration of joint angles (the pose) that fit the hip and ankle constraints. Different IK algorithms can be used for this and they will likely produce different looking results. However, to be usable at all for the methods described in this work, an IK algorithm must fulfill certain conditions:

- In each frame the IK adjustments must be performed from scratch while disregarding the IK adjusted pose in the previous frame. This avoids path dependencies, meaning that for a given instant in time, the same input parameters always results in the same output pose, regardless of the history of how the leg got into that pose.

- Rather than using the IK adjusted pose from the previous frame as input, the IK solver must use the original, unadjusted pose for the current frame, sampled from the original animations and blended according to the current blend weights. The IK solver must attempt to make the IK adjusted pose match the original pose as closely as possible, while still fitting the hip and ankle constraints. Usually, this entails making the adjusted joint angles match the original joint angles as closely as possible. What that means exactly can be interpreted differently by different algorithms.

- In order for the algorithm to not produce jittery or non-continuous results over time, even though it starts calculating a new adjusted pose from scratch in every frame, a requirement of the algorithm is that similar original poses will always also result in similar adjusted poses.

The implemented system uses two very simple inverse kinematics solutions. An analytical solution is used when there are only two bones between the hip and ankle joints (as it is the case for humans), while a primitive numerical solver is used when

there are more than two bones between hip and ankle. None of those IK solutions take any constraints of rotation angles into account, but simply attempt to find a pose as close to the pose in the original motion as possible.

Inverse kinematics are performed twice per frame per leg (see section 7.4.3). This is very efficient when using the analytical solver. For legs with more than two segments, it may be possible to use a better numerical solver with proper constraint handling to combine the two IK steps into one.

7.5 Starting and Stopping Walking

The methods described in this chapter lets each foot of the character take a step from any arbitrary position on the ground to any other. Once this flexibility is established, it can be used for more than just ensuring that the feet step correctly on uneven terrain. This section proposes a simple method to create procedural transitions between standing and walking or back from walking to standing.

There is an inherent incompatibility in the motion of a character standing still and a character walking or running. Both feet are standing firmly on the ground all the time when standing still, while in a walking or running motion, all the feet are never firmly grounded at the same time. For this reason, no transition between standing and walking can be properly made with traditional blending and cross-fading techniques alone. A traditional approach to solve that problem is to have example motions for the transitions between standing and walking, and back. That, however, has the draw-back that the character can only stop at the point in time during the walk cycle where the transitional motion begins, and is thus not able to stop at any given time.

7.5.1 Decoupled Leg Cycles

The method proposed here is based on automated transitions that do not require any transitional example motions. Instead, the timing of the individual leg cycles of the different legs are modified such that all the legs can come to a halt, one after the other, when the character is stopping walking. In the methods described above, the leg cycles of the legs are normally turning around in a synchronized fashion (see figure 7.1a on page 67). However, since the leg cycles are treated separately, their synchronization with the overall motion cycle can be broken when desired. When a character needs to come to a stop, each leg cycle can simply stop turning once it reaches its stance time, while the other leg cycles keep turning until they too reach their respective stance times.

In the implemented system, transitions between starting and stopping follow a few simple rules. If the length of the next step for a given leg is below a given threshold, the leg cycle is “parked” (put to a a stop) once it reaches its *stance time*. If the leg is already parked and the length of the next step is above a certain threshold, the leg cycle is “unparked” (starts turning around again) once the “would-be” leg cycle value passes the *stance time*. The “would-be” leg cycle value is the value it

would have had according to its synchronization with the overall motion cycles, if it had not been parked.

7.6 Results

The semi-procedural animation methods described in this chapter have been used on a variety of characters with different skeletons (multiple human and animal models), different amounts of legs (two and four legs), and different amounts of example motions (see chapter 8 for details). The stated framework goal of producing *flexible motion* has been found to be met. For all of the tested characters, the semi-procedural methods can produce straight, curved, sidestepping, and backwards motion, and anything in between; dynamic adaption to uneven terrain including arbitrary steps and slopes; and natural transitions between standing, walking, and running. Furthermore, the methods prevent any form of footskate.

In order to test the faithfulness of the adjusted motion to the example motions, the adjusted motion has been compared graphically to the example motions in conditions that exactly match the native conditions of the example motions (see chapter 8.4). In those performed comparisons, no measurable difference was spotted at all.

7.7 Summary of Semi-Procedural Animation

In order to make a character be able to walk or run in any direction on any uneven terrain, semi-procedural adjustments to the motion are performed in each frame to make the character take steps from and to arbitrary positions. For each leg, the footprints on the ground that a step should be taken from and to are calculated, with the *previous* footprint being known and the *next* footprint being continuously predicted based on the current velocity and rotational velocity of the character. The trajectory of the footbase from the *previous* to the *next* footprint is calculated based on the positions and alignments of the footprints, the motion of the character, and the blended result of the normalized footbase trajectories of the example motions. The foot alignment itself is calculated based on a simple model that determines which properties of the foot alignment from the original motion to preserve. Finally, the altitude of the hips of the character may be adjusted in order to keep the amount of bending of each leg as closely as possible to the pose in the original motion.

The described methods are procedural in that they are based on a model for legged locomotion with explicit concepts of legs, feet, taking steps etc. Yet, the produced motion is entirely data-driven, in that the more similar the runtime conditions (the velocity of the character and the terrain) are to the reference conditions of the original motion, the more similar the synthesized motion is to the original motion too. If the conditions match exactly, the synthesized motion matches the original motion exactly too.

Part III

Validation

Chapter 8

Results and Evaluation

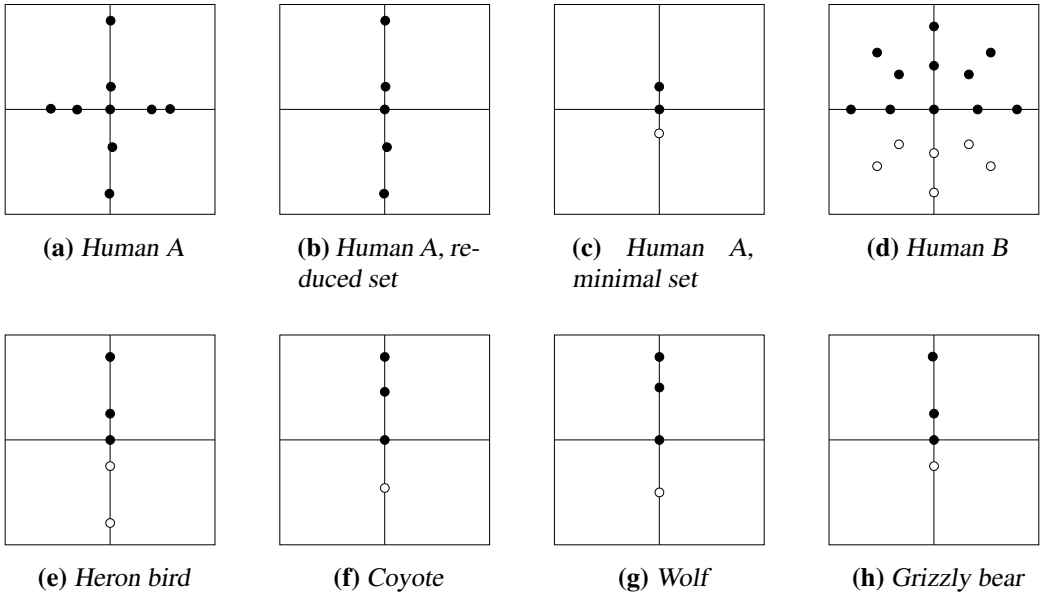
The results of the individual technical solutions are covered at the ends of each of the chapters 4 on page 27, 5 on page 41, 6 on page 48, and 7 on page 66, while this chapter presents the results of the work in this thesis at a more overall level and evaluates the framework as compared with the hypothesis and the stated framework goals (see chapter 1.2.1 on page 10).

The framework of methods for automated semi-procedural animation for character locomotion has been implemented and tested with a variety of characters with different skeletons (multiple human and animal models), and motion sets (see table 8.1 and figure 8.1). The lowest amount of examples tested was three (idle, forward, and backwards) while the highest amount tested is 17 (idle, and walking and running in eight directions each).

8.1 Genericness in Use of Characters and Styles

The implemented system has been tested with a variety of characters with two and four legs (see table 8.1). The system has been designed to handle characters with any number of legs, only limited by performance. In most respects the legs are treated separately from each other, which means that no specific order is assumed, and that any gait can be handled as long as each leg take exactly one step during a motion cycle. Certain parts of the semi-procedural adjustments are based on calculations that involve all the legs, such as the calculation of the hip height. In all those calculations, care has been taken to not assume anything about which legs are grounded and which are in mid-flight. Some, all, or none of the legs can be grounded simultaneously, which means that anything from normal walking to galloping, to hopping (on all legs) is supported. The tested gaits (human walking and running as well as quadrupedal trot and gallop) have all worked as expected with the system. Though not tested, the system is designed to also work fine for a creature with a single (hopping) leg.

Since the system performs no physical simulation, styles are not restricted to being realistic. The provided motions should however not have footskate to begin with, meaning that the feet should move with a constant velocity, relative to the



(a) *Human A* **(b)** *Human A, reduced set* **(c)** *Human A, minimal set* **(d)** *Human B*

(e) *Heron bird* **(f)** *Coyote* **(g)** *Wolf* **(h)** *Grizzly bear*

Figure 8.1: *Velocities of example motions of multiple characters. The origin marks zero velocity. Examples above the origin mark forward velocities; examples below mark backward velocities, and examples to either side mark sideways (strafing) velocities. Solid dots mark original example motions while white dots mark pseudo motions that have been obtained by playing back one of the original example motions in reverse, thus also obtaining the opposite velocity. This distinction, while interesting to note, is not relevant to the interpolation method though.*

Motion set	Examples	Description
Human A, full	9 (9)	idle, walk and run in four directions each
Human A, reduced	5 (5)	idle, walk and run forward and backwards
Human A, minimal	3 (2)	idle, walk forward and backwards
Human C	17 (11)	idle, walk and run in eight directions each
Heron bird	5 (3)	idle, walk and run forward and backwards
Coyote	4 (3)	idle, trot and gallop forward, trot backwards
Wolf	4 (3)	idle, trot and gallop forward, trot backwards
Grizzly bear	4 (3)	idle, trot and gallop forward, trot backwards

Table 8.1: *Parameterized motions in the experiments made for this work. In the middle row, the number of examples used in the interpolation is followed with a number in parenthesis marking the amount of original examples, as opposed to pseudo-examples, that are generated as a reverse of their respective opposite motions.*

character, while resting on the ground.

In its current implementation the system does not support a variable number of legs for one character. For example, it cannot be used for a creature that sometimes walk and two legs and other times crawl on four legs. This is more of a limitation in the implementation than in the general methods, although certain calculations would need to be slightly generalized in order to support a variable number of legs.

The implemented system assumes constant sizes of limbs and does not support stretchable limbs, or limbs that can expand or contract in other ways, such as hydraulic pistons. Again, this is mostly a limitation in the implemented system. Being able to handle non-constant limb sizes would be a matter of using a more flexible inverse kinematics system, and that system is already treated as a black box that can easily be exchanged with a different implementation that is more flexible.

None of the tested characters or motions were designed specifically with this locomotion system in mind. They were all created by other parties and for other purposes well before work had even begun on creating this system. As such, the system is shown to work fine with animated character that were made for actual use in computer games, and without requiring any modifications to be compatible with the system.

8.2 Flexibility of the Synthesized Motion

The stated framework goal of producing *flexible motion* has been found to be met. The semi-procedural methods described in this thesis produce flexible motions that lets the animated character take steps in arbitrary directions, regardless of which step directions were provided in the example motions. The character can also walk on any uneven terrain with arbitrary steps and slopes and have the feet land properly on the ground without footskate. Furthermore the system provides natural transitions between standing, walking, and running, with the exception that gaits with completely different foot fall patterns (such as quadrupedal trot and gallop) may sometimes show artifacts when interpolated between.

The character is controlled at a high level by the game logic, simply by specifying the position and orientation of the character in each frame. This means that the control scheme is completely open and for example not limited to specifying a speed and turning angle (although that is entirely possible if so desired). Rather, the character may move in any direction regardless of which way it is facing. Sidestepping, backwards walking, and movement in any other direction is possible, or the character might move in a straight line while twirling around all the while if so desired. Even in games that do not have sideways walking as part of the gameplay, the flexibility to take steps in arbitrary directions can still be used to greatly enhance the look of character movement, as discussed in chapter 3.

The system lets the character move over any kind of uneven terrain with arbitrary steps and slopes. The slope and height may be different for the two (or more) feet. The feet correctly land on the ground according to the slope of the ground at that spot. The concept of the footbase ensures that feet maintain the correct height

over the ground regardless of how the ground at runtime differs from the reference horizontal plane.

The foot trajectory logic is limited in that it only detects the ground at the positions of the footprints where the feet land. If any detail of the terrain is small enough not to be hit by a footprint, it will effectively be ignored by the system. This means for example that feet may pass right through a rock if the rock is small enough that none of the feet step on it. This limitation of the system was chosen for optimization reasons. If an efficient method can be implemented to query the height of the ground at many spots along the projected foot trajectory, it would be no major undertaking to make the feet better able to avoid intersecting the terrain. That being said, games today rarely have small details in the collision geometry, since it can generally cause many problems for the game logic, so the problem will likely rarely manifest itself in practice.

8.3 Ease of Use of the Framework

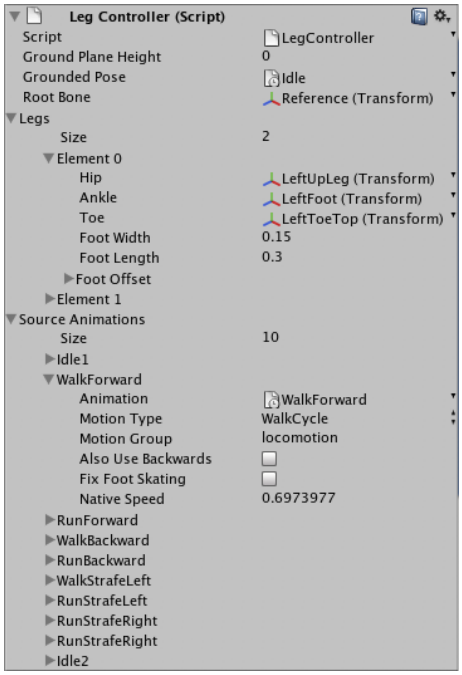
The implemented system is exceedingly simple to use relative to how complex it is behind the scenes. The user is presented with an interface with just a few pieces of data that need to be filled in (see figure 8.2).

In the LegController component, the user specifies a reference ground plane height and a reference animation clip where all feet are known to be grounded in the first frame. The user must specify the number of legs, and for each leg drag the hip bone, ankle bone and optionally a toe bone into dedicated slots. The foot width and height, and optionally an X and Z offset is specified, while a rectangle around the foot is drawn to reflect the typed values to make it easy to ensure that they are correct. The alignment is found automatically from the bones.

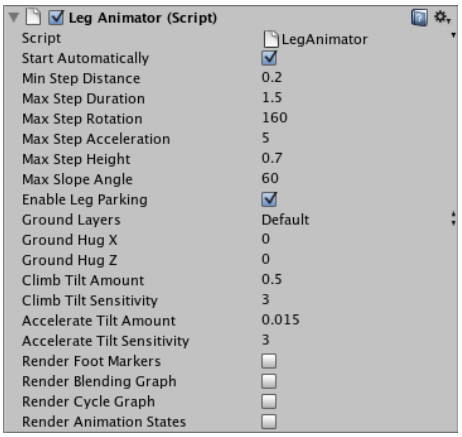
The user then specifies how many animations are to be controlled by the system, which should generally be all walk and run cycles as well as all grounded animations where all feet are on the ground. For each animation, the user needs to drag the animation clip into a dedicated slot, and specify whether that animation is a walk/run cycle or a grounded animation. The motion group name is specified. Each motion have a setting to enable it to be used as its own reverse equivalent also, so for example a forward walking motion can double as a backwards walking motion. There is also a setting to fix foot skating. This is a setting to fix footskate present in the *original* motion due to the motion being of poor quality, and not having all the feet move with the same velocity when in contact with the ground. The system will attempt to fix that if the option is enabled. The Native Speed should not be specified, as it is analyzed automatically.

After all settings in the LegController component has been filled in, a menu command can be invoked to start the automatic motion analysis. Filling in all the settings can take as little as five minutes for a person who have become comfortable with the interface of the system. The automatic analysis itself usually takes up to a few seconds, after which the system is ready to be used.

The settings in the LegAnimator component are less important to start with as



(a) *The LegController component. The user only need to specify a little data for each leg and for each animation. The data is used for the automated motion analysis. The "Native Speed" variable of each motion is special in that it is automatically analyzed, and is only shown in the interface for the user's convenience.*



(b) *The LegAnimator component. These variables are not used for the motion analysis but rather at runtime only, and may be altered during runtime. Sensible default values are provided to make the system typically work fine from the beginning without any adjustments needed.*

Figure 8.2: *The interfaces of the two components of the implemented locomotion system that the user setup the system with.*

they are not used for the motion analysis and have sensible default values. However, the values may be changed at any time to tweak the behavior of the locomotion system.

The implemented system has no scripting interface, as it simply continuously reads the position and alignment of the character and synthesizes motion appropriate for that velocity and rotational velocity. The different animation groups are invoked the same way that the user would normally invoke an animation to be played, or have a certain weight.

The reason that the locomotion system can have such a minimal interface is the amount of automated work performed combined with the genericness of the implemented solutions. Once the user has identified the important bones in the legs of the character and a few other trivial things, the input motions are analyzed completely automatically. There is no need to set up blend trees for the locomotion, as the scattered data interpolation method used is completely automated and has no parameters to tweak at all. (Manual blending control is only needed to control stylistic parameters.) The system has no control interface, as it just observes the changing position and alignment of the character and derives the velocity, rotational velocity, and everything else needed from that.

The minimal interface with most of the complexity of the system being hidden makes a tremendous difference in getting people to actually use it without any kind of training. At the time of writing, the implemented system is in use by a number of people currently developing games with it. Most have understood the system from a small provided tutorial alone, while a few have had some technical questions. Several people have shown me the system working with their own characters.

8.4 Faithfulness to the Original Motion

In order to test the faithfulness of the adjusted motion to the example motions, the adjusted motion has been compared graphically to the example motions in conditions that exactly match the native conditions of the example motions. For example, in order to test the faithfulness to a walking example motion, the character with the semi-procedural motion methods applied is set to run with the same velocity corresponding to the example motion, and on a flat plane. A copy of the character is walking alongside with the same velocity and on the same flat plane, but without semi-procedural methods applied. The silhouette of the character with semi-procedural methods is compared to the silhouette of the character without semi-procedural methods to test if the motions are identical (see figure 8.3). The silhouettes are compared on top of each other, so that even slight differences can easily be spotted, and this comparison is performed from multiple camera angles (cameras with orthographic projection are used so that perspective does not interfere). In those performed comparisons, no measurable difference was spotted at all. Only when the conditions are changed, such as changing the velocity or the slope of the plane, does the motion of the character with semi-procedural adjustments deviate from the the example motion, and this deviation is proportional to how different the

(a) *Walking with the ground at runtime being the same as the reference ground in the original motion. The adjusted motion is completely identical to the original motion. A slight offset has been given to the reference silhouette in order to make it not be completely hidden.*

(b) *Walking with the ground at runtime having a different slope than the reference ground in the original motion. The poses of the legs are adjusted enough to fit the new slope, but are otherwise the same as in the original motion.*

Figure 8.3: *Comparison of adjusted motion (foreground) with original motion (purple).*

conditions are.

8.5 Believability of the Synthesized Motion

The motion synthesized by the methods presented in this thesis generally look natural in most cases. See appendix A. Also see the video material and interactive demos accompanying this thesis to see the synthesized animation in motion.

It is an open question how the motion with semi-procedural adjustments compares to motion that has instead been synthesized from a much higher number of example motions. It is expected that a higher number of example motions used will produce more natural motion in a wider range of situations. However, it is clear from the provided comparisons that the motion certainly looks better with the procedural adjustments than without, all other things being equal.

The adjusted foot trajectories work well both for climbing slopes, steps, and arbitrary uneven terrain. The procedural foot alignment logic have shown to produce natural foot rolls both when walking up and down arbitrary slopes (or sideways along a slope) and when walking up and down steps of arbitrary height. The calculation of the hip height work well to maintain the bending of the legs as similar as possible as in the original motion; however it sometimes introduces undesired curves to the trajectory of the hips that may appear less smooth than the original hip trajectory, especially when walking or running on highly sloped surfaces. Some

Biped:	**0.25 ms per frame**	
1 biped:	100 -> 98 fps	50 -> 49 fps
10 bipeds:	100 -> 80 fps	50 -> 44 fps
Quadruped:	**0.55 ms per frame**	
1 quadruped:	100 -> 95 fps	50 -> 49 fps
10 quadrupeds:	100 -> 65 fps	50 -> 39 fps

Table 8.2: *Performance of the implemented locomotion system on a 2.4 GHz Intel Core 2 Duo. Performance was tested for a biped and a quadruped character. The milliseconds per frame per character was measured. Derived from this, the table gives examples of how much the frame-rate would drop when using the system with 1 and 10 characters respectively, given an initial frame-rate of 100 or 50.*

artifacts may also sometimes appear in the leg poses of characters with legs with more than two segments, due to the fact that the implemented system uses a very crude inverse kinematics solver for that, but in many cases even the crude solver produce natural-looking results.

The implemented feature to transition between standing and walking generally produces natural-looking results. Transitions between different gaits, such as walking and running also generally produce natural-looking results, with the exception that gaits with completely different foot fall patterns (such as quadrupedal trot and gallop) may sometimes show artifacts when interpolated between. Together with a system that can tilt characters based on acceleration, the movements when accelerating or coming to a stop look decent. The automated transitions are however not quite up to the quality that can be achieved with using dedicated example motions for starting and stopping, but on the other hand the automated transitions are more flexible.

8.6 Efficiency

The locomotion system has been implemented using C# scripting in the Unity game engine. On a MacBook Pro 2.4 GHz Intel Core 2 Duo, the locomotion system can be used on dozens of characters at high frame rates. Performance was tested with a biped human character and with a quadrupled wolf character (see table 8.2). The quadruped takes significantly longer computation time per frame for two reasons. It has twice as many legs, which doubles many of the calculations, and each leg for the specific tested quadruped has three segments per leg rather than two, which means it must use the slower numeric inverse kinematics solver, while the biped can use the fast analytical solver.

High performance was not the primary concern while developing the system and there are many optimizations that could be done to optimize it. The implemented Gradient Band interpolation algorithm performs in $O(n^2)$, where n is the number of example motions, but it can be reduced to constant time by using a look-up ta-

ble over which examples have non-zero influence in which areas of the parameter space. A numeric inverse kinematics solver with better performance could be implemented. Two raycasts are used per foot in every frame, but doing it less frequently might still produce acceptable results.

Worth noting is that the semi-procedural adjustments can be turned gracefully on and off for a character without causing any popping or other non-continuous artifacts. This means that they can be applied to only the characters closets to the camera at any given point in time, while character further away, where footskate etc. are not as visible, can have the system turned off and rely on regular blended motion.

Chapter 9

Conclusion and Future Work

> The future of technology for animated characters in interactive systems lies as a hybrid of [example-based approaches] and algorithmic approaches. (Gleicher, 2008)

9.1 Conclusion and Discussion

The goal of this work has been to create a framework of techniques for automated semi-procedural animation that can be used to obtain flexible and believable character locomotion, while the amount of needed example motions and the time and expertise spent on setting up the animation framework can be kept to a minimum. Chapter 1 listed a set of stated framework goals to give an overview of the multiple challenges that should all be addressed by the framework, and the implemented system has been evaluated with regard to each of those goals in chapter 8. In general, the system has been found to produce motion that is *flexible* to fit the environment, *faithful* to the original motion, and *believable* in its adjustments. At the same time the system is *generic* in which characters it can be used with, *easy to use*, and *efficient* enough to be used with multiple characters.

Though many techniques used in this thesis are similar to what others have done before, new methods have been proposed to solve several motion synthesis problems in novel ways.

Chapter 4 introduced the concept of a footbase, which is a single combined heel and toe constraint. Its usefulness stems from the fact that it can retain the important information about the alignment of a foot relative to the ground, no matter if both the alignment of the foot and of the ground is different at runtime than in the original reference motion. The use of a footbase constraint rather than individual heel and toe constraints can significantly simplify the task of generating adjusted motion of feet that must take steps on uneven terrain without penetrating the ground, incorrectly hovering over it, or displaying footskate artifacts.

Chapter 5 advocated the concept of animation groups. Animation groups are similar to the concept of blend trees, and they are used in this work specifically to separate the manual control of stylistic parameters from the automated control

of physical parameters. Animation groups have the ability to generate controllable blended motion in a way that is more predictable than when using a single multidimensional parameter space.

Chapter 6 introduced a new method for scattered data interpolation that has many desirable properties and can be used for getting intuitively natural blend weights for interpolating between motions with different directions and magnitudes (such as velocities) without requiring any tweaking of variables from the user.

Chapter 7 presented many different methods for procedural adjustment of motion that all have in common that they synthesize motion that is faithful to the original motion, while being highly flexible. The combined result is unpreseded flexible locomotion in how few example motions are required to produce motions with steps in arbitrary directions on arbitrary terrain, when also considering the range of different characters and styles of motion that the framework can be used for.

Of the methods for procedural adjustment of motion, two novel techniques are worth pointing out in particular. One is the calculation of a *supporting ground height* in section 7.3.6, which can be used to create a natural looking overall trajectory for a character, that produces motion with a good sense of weight, since it gives highest priority to preserving the amount of bending for the legs that are currently supporting the character. Similar observations have been done before, but to my knowledge not in a way that works for any number of legs and gait styles, including, but not limited to walking or running bipeds or trotting or galloping quadrupeds. Another novel technique is the calculation of natural-looking foot alignments in section 7.4.3. The method distinguishes itself from prior described techniques for foot alignment in that it again works for just about any characters and gait styles.

One technique that has shown not to work optimally in all cases is the method for hip adjustment described in section 7.4.2. While the method does a good job of keeping the bending of the legs as similar as possible to the original motion, it sometimes introduces disagreeable artifacts in the trajectories of the hips, especially when walking or running on highly sloped surfaces.This method needs to be reconsidered with higher priority given to keeping the hip trajectories smooth.

Besides the individual technical contributions of the various methods, the framework as a whole has been designed in every respect to be as automated as possible and require as little input as possible from the user. The fact that many people have started using the implemented framework with their own characters without any training and with only a short tutorial as introduction to the framework shows the merit of finding solutions to problems that are general enough not to only require a bare minimum of user intervention.

Part of the motivation for this work was to create methods for improved character animation that is accessible and cheap enough to use that is can be used not only by big game studios with easy access to large amounts of motion-capture data or animation in other forms, but also for small developers with tight budgets and minimal resources. I believe the methods presented in this thesis go along way towards that goal in the specific area of character locomotion, though not without areas left for improvement.

9.2 Future Work

The possibilities of the presented approach as well as the limitations of the current implementation point towards many areas of future work, both in extending the framework for character locomotion and in going beyond that.

9.2.1 Extended Locomotion Features

One of the major strengths of the presented methods is the fact that completely flexible motion can be obtained with a minimal amount of example motions, and as more example motions are added, the quality of the motion simply increases. A high number of example motions are never needed to get the desired flexibility, but if they are present, they will be taken full advantage of. However, a big limitation of the implemented system is that it only uses example motions with constant velocity on a flat horizontal plane. Supporting example motions with a non-zero turning angle, slope, and step height could improve upon the already flexible approach and give the user an even higher roof for the maximal possible quality if enough resources are available.

Supporting example motions with non-horizontal velocities could also open the door towards using the system for crawling on ladders and walls. However, before that is possible, the methods would need to be generalized to handle a variable number of legs so that walking on two legs and crawling ladders with both legs and hands can co-exist in the model.

A simple addition to the methods would be a way to specify valid and invalid footprint areas. The current system already attempts to avoid stepping at the edge of a ledge, but with user-definable areas in the environment to avoid or pursue stepping on, seemingly more clever behavior could be shown, such as stepping on stepping stones in a lake to cross it dry-shod.

9.2.2 Increased User Guidance

One goal of the described methods is to make systems possible that are so easy to use and require so few resources that even small game studios can make use of it. Such small studios may have limited budgets, and the automated analysis can save time that would otherwise have been needed for animators to manually annotate the provided animations with precise velocities and keytimes for each leg, as well as the time needed to learn how to do that. However, such small studios may also only have animators with mediocre skills (or the people creating the animations may not be dedicated animators at all), and thus may not always be able to create animations of consistent high quality. One possible measure that can be taken to oblige this issue is to extent the automated analysis to attempt to detect when there are possible problems with the provided animations, and in those cases communicate those problems to the user in a way as clear as possible. Furthermore, the analysis process could let the user correct any possible mistaken results of the analysis in a simple

interactive fashion.

9.2.3 Semi-Procedural Modules

Can the methods presented here be generalized to work for broader classes of motions, besides locomotion? I think that this thesis shows that designing semi-procedural methods that are highly generic and flexible while still able to produce natural and believable motion on scarce input data is a matter of making just the right assumptions, in order to create a model that assumes neither too little nor too much. This thesis present methods that are based on a model of legged locomotion and as such cannot be extended to work for other types of motions too. However, models can be made for other types of motions based on the same general philosophy.

A simple semi-procedural module that has been used in a lot of games for a long time is a head-look adjustment. A character that is animated based on classic synthesis of example motions can have the direction the head is looking in controlled procedurally on top of the blended motion. By letting the user identify bones for the eyes, the head, and the upper body as well as some quickness values for each, the character could be made to look in any direction in a natural-looking way with the eyes looking quickly, the head turning slower, and finally the upper body slowly turning a bit as well. Other modules could be made for punching, pointing, or picking in arbitrary directions based on just one or a few example motions, and methods such as this has in fact been described in previous work already. Many modules are much simpler to design than the multiple-constraint case of locomotion; yet not simple enough that they are trivial for many users to implement on their own. Having a library of such semi-procedural motion adjustment modules at his disposal, even small developers with limited resources could be equipped to create games with animation much more flexible and natural-looking than typically seen today.

Bibliography

Allen, B., Curless, B., and Popović, Z. (2002). Articulated body deformation from range scan data. *ACM Trans. Graph.*, 21(3):612–619.

Amidror, I. (2002). Scattered data interpolation methods for electronic imaging systems: a survey. *Journal of Electronic Imaging*, 11(2):157–176.

Bruderlin, A. and Calvert, T. (1993). Interactive animation of personalized human locomotion. In *Proceedings of Graphics Interface '93*, pages 17–23, Toronto, Ontario, Canada. Canadian Information Processing Society.

Bruderlin, A. and Calvert, T. (1996). Knowledge-driven, interactive animation of human running. In *GI '96: Proceedings of the conference on Graphics interface '96*, pages 213–221, Toronto, Ont., Canada, Canada. Canadian Information Processing Society.

Bruderlin, A. and Williams, L. (1995). Motion signal processing. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 97–104, New York, NY, USA. ACM.

Choi, M. G., Lee, J., and Shin, S. Y. (2003). Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Trans. Graph.*, 22(2):182–203.

Fêdor, M. (2003). Application of inverse kinematics for skeleton manipulation in real-time. In *SCCG '03: Proceedings of the 19th spring conference on Computer graphics*, pages 203–212, New York, NY, USA. ACM.

Forsyth, T. (2004). How to walk - a complex tutorial on a seemingly simple subject. Powerpoint slides. `http://home.comcast.net/~tom_forsyth/papers/papers.html`; accessed February 16, 2009.

Gleicher, M. (1998). Retargetting motion to new characters. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 33–42, New York, NY, USA. ACM.

Gleicher, M. (2001a). Comparing constraint-based motion editing methods. *Graph. Models*, 63(2):107–134.

Gleicher, M. (2001b). Motion path editing. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 195–202, New York, NY, USA. ACM.

Gleicher, M. (2008). More motion capture in games - can we make example-based approaches scale? In Egges, A., Kamphuis, A., and Overmars, M., editors, *Motion in Games*, Lecture Notes in Computer Science.

Heck, R. and Gleicher, M. (2007). Parametric motion graphs. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 129–136, New York, NY, USA. ACM.

kai Chung, S. and Hahn, J. K. (1999). Animation of human walking in virtual environments. In *CA '99: Proceedings of the Computer Animation*, pages 4–15, Washington, DC, USA. IEEE Computer Society.

Komura, T., Leung, H., and Kuffner, J. (2004). Animating reactive motions for biped locomotion. In *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 32–40, New York, NY, USA. ACM.

Kovar, L. and Gleicher, M. (2003). Flexible automatic motion blending with registration curves. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 214–224, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

Kovar, L. and Gleicher, M. (2004). Automated extraction and parameterization of motions in large data sets. pages 559–568.

Kovar, L., Schreiner, J., and Gleicher, M. (2002). Footskate cleanup for motion capture editing. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 97–104, New York, NY, USA. ACM.

Kwon, T. and Shin, S. Y. (2005). Motion modeling for on-line locomotion synthesis. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 29–38, New York, NY, USA. ACM.

Lee, J. and Shin, S. Y. (1999). A hierarchical approach to interactive motion editing for human-like figures. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 39–48, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

Meredith, M. and Maddock, S. (2005). Adapting motion capture data using weighted real-time inverse kinematics. *Comput. Entertain.*, 3(1):5–5.

Park, S. I., Shin, H. J., and Shin, S. Y. (2002). On-line locomotion generation based on motion blending. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 105–111, New York, NY, USA. ACM.

Rose, C. F., Bodenheimer, B., and Cohen, M. F. (1998). Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18:32–40.

Rose, C. F., Sloan, P.-P. J., and Cohen, M. F. (2001). Artist-directed inverse-kinematics using radial basis function interpolation. *Computer Graphics Forum*, 20(3):239–250.

Semančík, J., Pelikán, J., and Žára, J. (2004). Interactive synthesis of constrained motion from example movements. In *Proceedings of the 4th IASTED International Conference on VISUALIZATION, IMAGING, AND IMAGE PROCESSING*, pages 878–883, Calgary, AB, Canada. Acta Press.

Sloan, P.-P. J., Rose, C. F., and Cohen, M. F. (2001). Shape by example. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 135–143, New York, NY, USA. ACM.

Sok, K. W., Kim, M., and Lee, J. (2007). Simulating biped behaviors from human motion data. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 107, New York, NY, USA. ACM.

Srinivasan, M., Metoyer, R. A., and Mortensen, E. N. (2005). Controllable real-time locomotion using mobility maps. In *GI '05: Proceedings of Graphics Interface 2005*, pages 51–59, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada. Canadian Human-Computer Communications Society.

Sun, H. C. and Metaxas, D. N. (2001). Automating gait generation. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 261–270, New York, NY, USA. ACM.

Torkos, N. and van de Panne, M. (1998). Footprint-based quadruped motion synthesis. In *Graphics Interface*, pages 151–160.

Unuma, M., Anjyo, K., and Takeuchi, R. (1995). Fourier principles for emotion-based human figure animation. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 91–96, New York, NY, USA. ACM.

van de Panne, M. (1997). From footprints to animation. *Computer Graphics Forum*, 16(4):211–223.

Wiley, D. J. and Hahn, J. K. (1997). Interpolation synthesis of articulated figure motion. *Computer Graphics and Applications, IEEE*, 17(6):39–45.

Witkin, A. and Popovic, Z. (1995). Motion warping. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 105–108, New York, NY, USA. ACM.

Wrotek, P., Jenkins, O. C., and McGuire, M. (2006). Dynamo: dynamic, data-driven character control with adjustable balance. In *Sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames*, pages 61–70, New York, NY, USA. ACM.

Yamane, K., Kuffner, J. J., and Hodgins, J. K. (2004). Synthesizing animations of human manipulation tasks. pages 532–539.

Yin, K., Coros, S., Beaudoin, P., and van de Panne, M. (2008). Continuation methods for adapting simulated skills. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–7, New York, NY, USA. ACM.

Yin, K., Loken, K., and van de Panne, M. (2007). Simbicon: simple biped locomotion control. page 105.

Appendix A

Comparison of Procedural Adjustments

Figure A.1: *Comparison of synthesized motion of human character with semi-procedural adjustments (foreground) and without (background).*

Figure A.2: *Comparison of synthesized motion of grizzly bear character with semi-procedural adjustments (foreground) and without (background).*

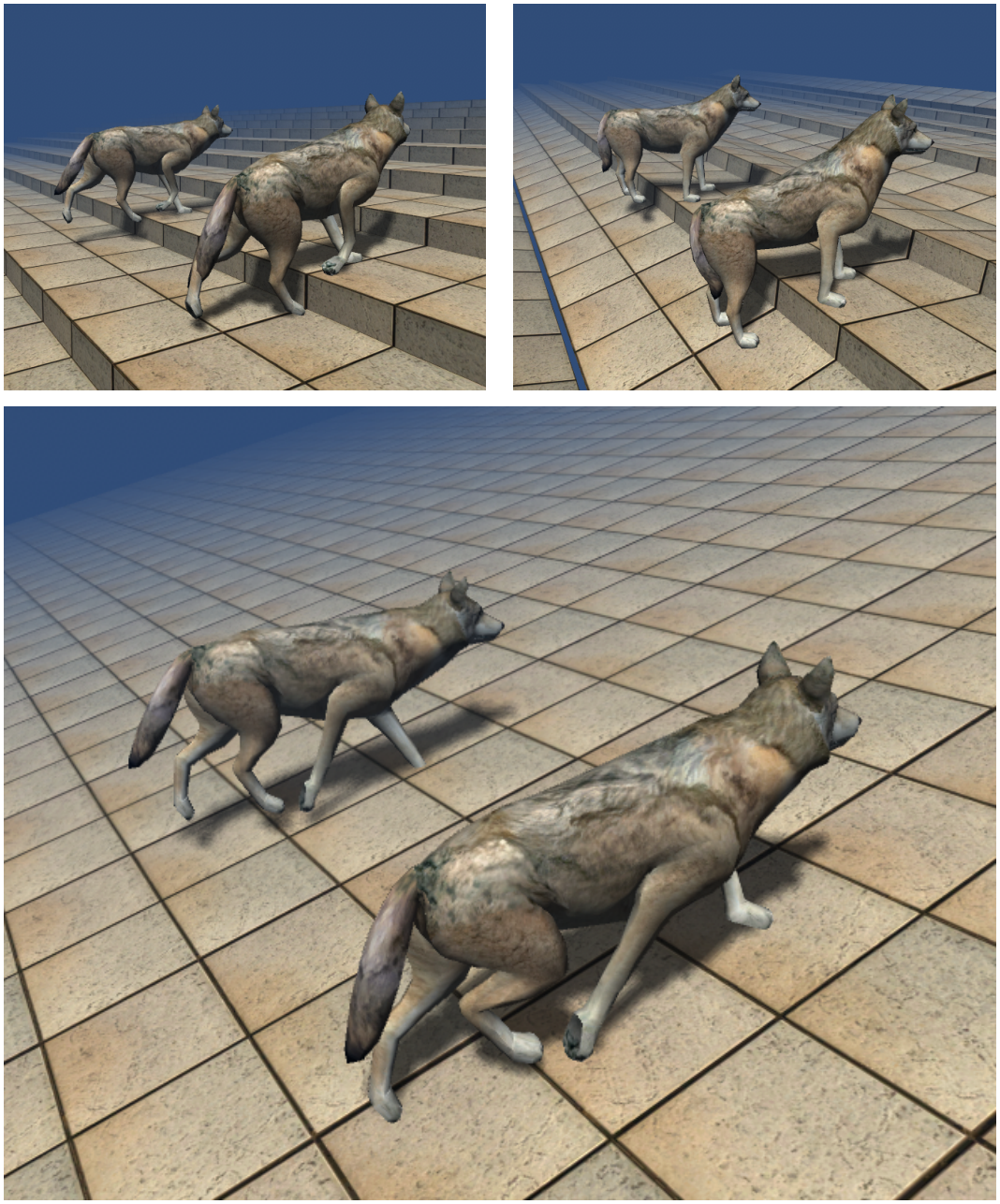


Figure A.3: *Comparison of synthesized motion of wolf character with semi-procedural adjustments (foreground) and without (background).*