

Thesis no: MECS-2015-03



Performance of Physics-Driven Procedural Animation of Character Locomotion For Bipedal and Quadrupedal Gait

Jarl Larsson

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Engineering: Game and Software Engineering. The thesis is equivalent to 20 weeks of full-time studies.

Contact Information:

Author:

Jarl Larsson

E-mail: jarl.larsson@gmail.com

University advisors:

Ph.D. Veronica Sundstedt

Ph.D. Martin Fredriksson

Department of Creative Technologies

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Context. Animation of character locomotion is an important part of computer animation and games. It is a vital aspect in achieving believable behaviour and articulation for virtual characters. For games there is also often a need for supporting real-time reactive behaviour in an animation as a response to direct or indirect user interaction, which have given rise to procedural solutions to generate animation of locomotion.

Objectives. In this thesis the performance aspects for procedurally generating animation of locomotion within real-time constraints is evaluated, for bipeds and quadrupeds, and for simulations of several characters. A general pose-driven feedback algorithm for physics-driven character locomotion is implemented for this purpose.

Methods. The execution time of the locomotion algorithm is evaluated using an automated experiment process, in which real-time gait simulations of incrementing character population count are instantiated and measured, for the bipedal and quadrupedal gaits. The simulations are measured for both serial and parallel executions of the locomotion algorithm.

Results. Simulations of up to and including 100 characters are performance measured providing an overview of the slowdown rate when increasing the character count in the simulations, as well as the performance relations between bipeds and quadrupeds.

Conclusions. The experiment concludes that the evaluated algorithm on its own exhibits a relatively small performance impact that scales almost linearly for the evaluated population sizes. Due to the relatively low performance impacts it is thus also concluded that for future experiments a broader measurement of the locomotion algorithm that includes and compares different physics solvers is of interest.

Keywords: Procedural animation, Interactive games, Multithreading, Software performance

Acknowledgements

Warm thanks to my family and friends, near and dear; for all your support and encouragement! Big thanks to my supervisors, Veronica Sundstedt and Martin Fredriksson, for guiding me through this project. Lastly, a special thanks to Michiel van de Panne for the insightful answers to my questions about procedural locomotion.

Contents

Abstract	i
1 Introduction	1
1.1 Procedural Articulation	2
1.2 Procedural Terrestrial Locomotion	4
1.3 Problem Statement	6
1.3.1 Aims and Objectives	7
1.4 Research Question	8
1.5 Method Introduction	8
1.6 Contribution	8
1.7 Thesis Outline	8
2 Background and Related Work	10
2.1 Locomotion	11
2.1.1 Believability in Animation	11
2.1.2 Terrestrial Locomotion Terminology	11
2.2 Dynamics-Driven Procedural Locomotion	12
2.2.1 Inverted Pendulum Methods	14
2.2.2 Muscle-Driven Methods	14
2.2.3 Pose-Driven Feedback Methods	15
2.3 Parallel Execution of Locomotion Algorithm	16
2.4 Performance in Parallel Algorithms	16
2.5 Summary	17
3 Implementation	18
3.1 Implemented Algorithm	18
3.1.1 Gait Player	21
3.1.2 Virtual Forces	21
3.1.3 Leg Frame and Torque Feedback	22
3.1.4 Legs	23
3.1.5 Gravity Compensation	24
3.1.6 Foot Placement	25
3.1.7 Proportional-Derivative Controllers	26
3.2 Constrained Rigid Bodies using BulletPhysics	26

3.3	Optimization	26
3.4	Controller Logic	28
3.5	Summary	30
4	Method	32
4.1	Scenarios and Parameters	32
4.2	Technology	35
4.3	Validity Threats	36
4.4	Summary	37
5	Results	38
5.1	Observations	38
5.1.1	Measurements	38
6	Analysis	42
6.1	Serial Results Analysis	42
6.2	Parallel Results Analysis	44
6.3	Trends and Relations	44
6.4	Summary	46
7	Conclusions and Future Work	48
7.1	Thesis Summary	48
7.2	Method and Implementation Discussion	49
7.3	Conclusions	50
7.4	Contribution and Comparisons	51
7.5	Future Work	51
	References	53

Character locomotion is a common subject within the domains of computer animation and games. Locomotion is the description of how propulsive movement is achieved by a body, for animals this translates to the resulting movement pattern of their limbs and frames. Terrestrial animal locomotion, more specifically, is the way in which an animal self-propel on land by, for example, walking or running. This kind of locomotion and its subsequent animation is common for characters that moves around in modern computer games.

Characters in computer games are often displayed as a collection of connected polygons and vertices, and thus needs some form of high-level abstracted control layer to express motions on a per-leg or per-arm level. Animation of polygon based characters in computer games are thus often realized and simplified by means of *morphing* [30, 35] or *skeletal animation* [39]. Both are control techniques used to transform the multitude of vertices of a 3D model, frame-by-frame, in order to animate it. Both techniques interpolates between keyframes of animation data.

Morphing needs transformed copies of the model for each pose to morph to, while skeletal animation lets the animator control an approximation of the character to be animated. This approximation, a hierarchy of segments, is called a *skeleton* (it can be likened to the common poseable wooden mannequins used by artists). By approximating the character to be animated, the animator will not have to displace each vertex manually. The segments in the skeleton are referred to as *bones*. Each bone describes a limb transformation, as a function of time, for the character. Each vertex in the character model belongs to one or several bones (with weighted priorities) and will follow their transformations. Skeletal animation and morphing can be combined in various ways as well, depending on the application.

By using a skeleton, one can thus control separate limbs on a character on a high level. Animation of locomotion can then be done on a per-leg basis.

Even though skeletal animation might lessen the amount of parameters to control for transforming and animating character geometry, the full range of motions exerted by one character alone can become a lot. A computer game featuring a rich amount of characters with varying skeleton layouts for animation may thus require vast amounts of manual animation work for only various forms of locomo-

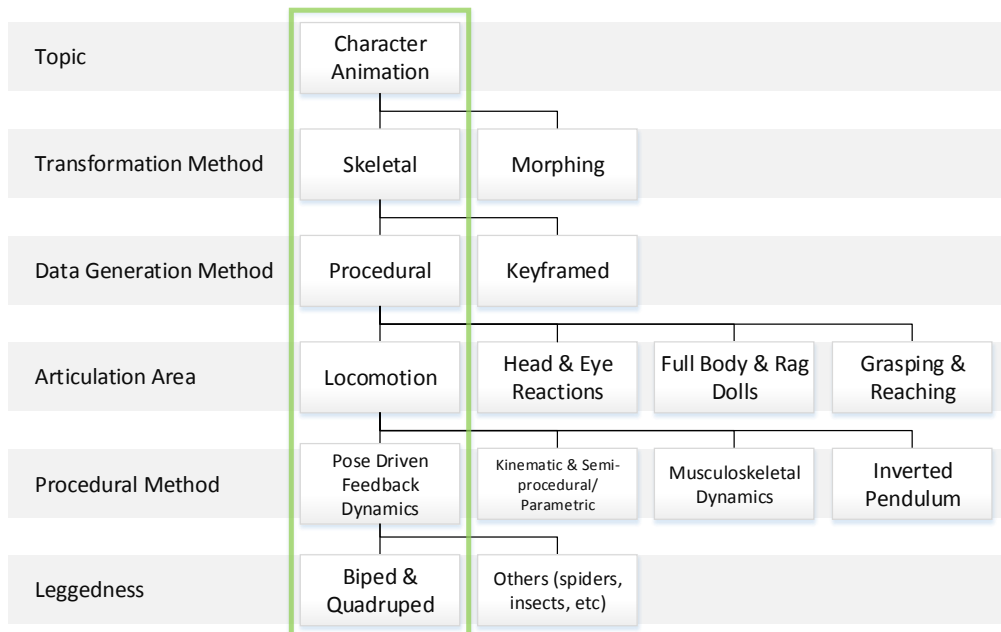


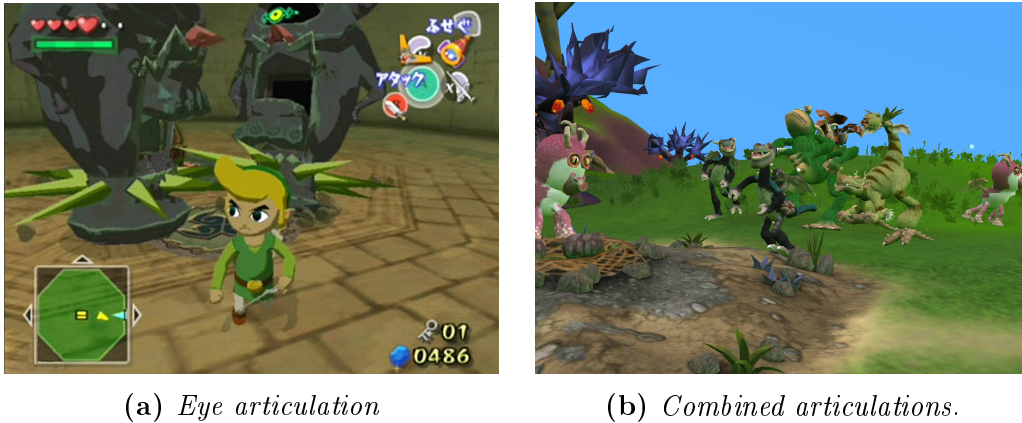
Figure 1.1: Flowchart of domains related to the topic of character animation. Represented as an hierarchical structure, the chart is read from top to bottom and visualizes the various ways in which a specific domain can branch. Domains for leaf nodes to the right are not focused upon in this thesis.

tion alone [21, 39]. When these game characters are expected to work in a large variety of environments and situations one more or less have to have some form of *procedural animation* component [46]. Not only to alleviate the animators of an exaggerated amount of manual keyframe animation labour, but also to minimize the memory footprint of the application [39].

Procedural animation solutions can manifest themselves in many different forms depending on the problem and platform. Figure 1.1 presents a hierarchical view of various domains within the topic of character animation and how this thesis will present them. Here procedural animation is separated from keyframed animation. The following section (Section 1.1) will introduce the various combinable ways a character can express itself through motions; its *articulations* (Figure 1.1), of which one is locomotion.

1.1 Procedural Articulation

An animated character can express a range of articulations, and the realization of these are commonly divided into separate solutions. Expressing emotion for



(a) Eye articulation

(b) Combined articulations.

Figure 1.2: Various forms of articulation used to express emotion and thoughts. In the game *The Legend of Zelda: The Wind Waker*, eye movement (a) is used to convey what a character is focusing on. *Spore* combined procedural and artist made animations to convey a range of emotions (b).

Source: (a) *The Legend of Zelda: The Wind Waker*: ©2003 Nintendo
 (b) *Spore*: ©2008 Maxis (Developer), Electronic Arts (Publisher)

digital characters is a vast problem area [5], and succeeding relies on combining various techniques for conveying thoughts and emotions effectively.

Reactive behaviour for characters in form of procedural animation of the head and eyes can be used to convey what a character is currently observing [5] (or trying to observe), or what it is trying to ignore. Grasping, reaching and locomotion can be used as a means to achieve or reach a goal. Such intentions can be made clearer by for example combining it with having the head and eyes look towards the goal. Examples of character eye movement as well as a combination of articulation techniques can be seen in Figure 1.2.

The underlying methods to procedurally achieve and animate various forms of articulation overlaps somewhat even though the end usage domains differ.

In modern games, various layering techniques are often used to combine multiple shorter animation clips of skeletal animation together [39], in order to decrease the total amount of animation data. One example is the *Uncharted* series by developer *Naughty Dog* [17,37]. Various corrective systems might be necessary as well for repositioning and realigning already animated bones for a character based on the surrounding environment [20]. Animation systems that changes already existing animation data to work in a new context, are usually referred to as *parametric* [36,48,55] or *semi procedural* [21], see Figure 1.1. These are sometimes referred to as being *kinematic* methods.

Within the domain of character articulation for locomotion, semi-procedural solutions can be used to avoid locomotion artifacts [59] (such as ill-timed foot strikes or wrongly placed feet) on an already animated character. However, in order to generate locomotion procedurally from scratch, without already made

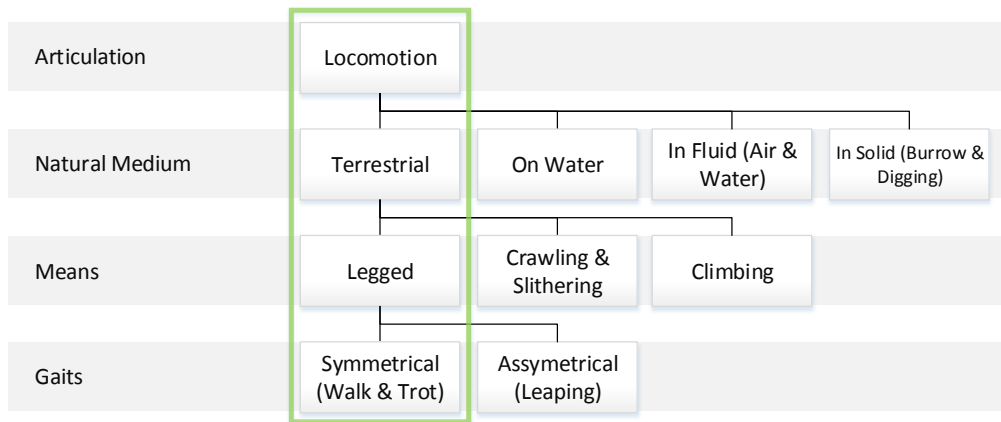


Figure 1.3: Flowchart showcasing the various forms of animal locomotion and how they relate. It describes the context (terrestrial legged locomotion) in which gaits are used in this thesis. Domains for leaf nodes to the right are not focused upon in this thesis.

animation data, we need to first look at what kind of locomotion is to be generated.

The way in which locomotion is expressed depends on the medium to which the characters relies for propulsion (see Figure 1.3). Movement in media, such as flight in air, swimming in water or digging through a solid will result in other environmental dependencies and results compared to movement on media. Movement on media in turn will differ for solids and fluids. There exist several means of propulsion on solids, these can be narrowed down to the use or non-use of legs (see Figure 1.3). Locomotion on a solid is known as *terrestrial locomotion*.

The following section (Section 1.2) presents the various techniques that exist for generating terrestrial locomotion procedurally without animation data.

1.2 Procedural Terrestrial Locomotion

The game *Spore* by *Maxis* presented a delicate animation problem of having to animate user-created creatures of varying anatomies and skeleton topologies. It used several kinematic systems that, when combined, would let a legged creature move about in uneven terrain, with a procedural animation solution that generated new frames of animation and transformations for bones based on movement velocities and terrain features [19]. In summation, this system presented one possible solution for procedurally solving and generating animation for terrestrial locomotion.

The specifics of how an animal moves its legs during terrestrial locomotion

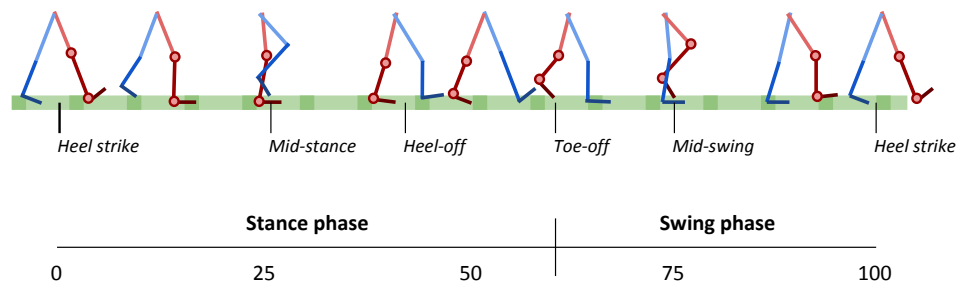


Figure 1.4: Gait analysis of human walk cycle, using classical terminology. The phases are expressed in percentage of the total gait cycle. The figure is adapted from previously published gait data [56].

in relation to its speed is known as *gait* [1, p. 14]. Gaits are patterns defining the locomotive cycle needed to achieve a certain movement speed for a certain animal.

Gait patterns can be expressed for a varying amount of limbs and styles of locomotion. An example of a gait analysis of the human walk cycle can be seen in Figure 1.4. In Spore, gait patterns are generated based on the amount of feet and their layout. This pattern then serves as the basis for the overall locomotion animation [19].

For procedurally animating the gait of a character, there exist several methods, as shown in Figure 1.1. These methods mainly differ in the amount of data and computation that are needed to generate the final motion. A procedural *kinematics-driven* animation system directly manipulates limb positions and angles to generate procedural motion. *Dynamics-driven* solutions, also known as *physics-driven*, estimates forces and torques needed for a physics solver to generate the final motions. This thesis focuses on physically simulated limbs that propels the character forward. In games, character physics can effectively be abstracted further, regarding a character as just a single rigid body or similar, for locomotion [47]. Though there are examples of more complex simulation solutions such as *Euphoria*¹ by *NaturalMotion*. Upcoming games (as of writing) such as *Uncharted 4*, are showing a trend towards a more common usage of more complex abstractions, with full body physics combined with motion capture data².

For the purely procedural methods; dynamics-driven procedural solutions are in general regarded to yield results with more physical realism compared to kinematics-driven pure procedural methods, which tend to suffer from stiffness in motion. However, a kinematics-driven animation system has the potential of pro-

¹www.naturalmotion.com/, Accessed 2015-01-31

²www.gameinformer.com/b/features/archive/2015/01/23/

[how-uncharted-4-is-taking-game-technology-to-the-next-level.aspx](http://www.gameinformer.com/b/features/archive/2015/01/23/how-uncharted-4-is-taking-game-technology-to-the-next-level.aspx), Accessed 2015-01-31

viding a smaller performance impact compared to dynamics-driven methods [22].

One dynamics-driven locomotion method is the *inverted pendulum* [28] model which abstracts away the knees from the character armature and calculates foot placements to balance the armature approximation. Positions and angles for the knees and feet are then free to be generated either kinematically or solved by a physics solver [53].

For better biological realism, one can simulate the muscles driving the movement of each bone in a character, using musculoskeletal dynamics [57].

Pose-driven dynamics, is a third dynamics-driven option with roots in the fields of control logic- and robotics theory. In pose-driven methods, each joint is simulated just like in the musculoskeletal methods, but in a simpler fashion, more akin to those of a robot. The simulation tries to achieve precomputed poses, taking physics into consideration.

Feedback mechanisms, such as foot-to-ground collision can be used to further control a dynamics-driven simulation appropriately.

All these various methods of dynamics-driven procedural animation of locomotion differ in resulting visual quality and performance impact. In regards to performance and in the context of a game with real-time aspects, the need to update a multitude of limbs has the possibility to become computationally expensive. Though these dynamics-driven methods are presented as capable of real-time simulation, there exists a lack of measurements of how such algorithms scale performance-wise when simulating multiple characters.

1.3 Problem Statement

The amount of academic research available on the performance impact of dynamics-driven locomotion algorithms is lacking. Though intelligence in this area might exist internally within the games- and software industry.

Articles presenting dynamics-driven procedural animation methods for locomotion focuses on the animation quality aspects. Some performance information is briefly mentioned for some of these works, to prove that they are able of real-time execution [8, 9, 13, 14, 27, 57].

Another aspect lacking is the performance of said algorithms for more than one character, although it has been reported as viable within real-time speeds [26, 49, 51] in single-threaded environments.

For dynamics-driven locomotion algorithms, there thus exist a gap for what kind of execution times can be expected for such an algorithm for bipeds and quadrupeds, including the performance effects of increasing the amount of simulated characters.

1.3.1 Aims and Objectives

Developing and investigating a solution for procedural animation might provide insights of its possibilities performance-wise as well as provide a platform for simulating the locomotive movement of several creatures in games and similar real-time visualization oriented software, with the possibility of richer environments [34].

Knowledge of performance impact in games is necessary as games needs to execute in real-time speeds to convey a sense of continuity to the user and not hamper the user's ability to interact with the software [38]. Thus it becomes necessary to identify what trends can be observed on the performance of a procedural locomotion system handling the cyclical feet- and leg movements of groups of characters, in a game-like setting (in this thesis: an application capable of rendering the simulated result in 3D, in real-time).

Pose- and feedback-driven real-time simulation methods for locomotion, offer physically simulated movement which can respond to external forces, without the need to simulate full muscle movement. It might thus be a strong contender for an interactive application or game.

The aim of this thesis is to extend the knowledge available for performance impact of procedural locomotion algorithms. This thesis will focus on one: a pose- and feedback driven algorithm. It will furthermore focus on simulation of bipedal and quadrupedal characters using such an algorithm. As the current generation of PCs and video game consoles (*SONY PS4* and *Microsoft Xbox One*) provide viable platforms for executing parallel algorithms, the thesis will provide an overview of how performance scales between serial- and parallel execution of the algorithm as well.

The thesis aims to fulfil the following objectives:

- Implement a pose- and feedback-driven locomotion algorithm for bipeds and quadrupeds, based on existing work.
- Present the performance impact of the implemented pose- and feedback-driven locomotion algorithm.
- Show how the performance of the algorithm is affected when increasing the amount of simulated characters.
- Present performance relation between bipedal- and quadrupedal characters simulated by the algorithm.
- Present performance relation between single-threaded and multi-threaded executions of the algorithm.

1.4 Research Question

Given a physics-driven procedural locomotion system, which uses a pose- and feedback-driven simulation method: what real-time performance relation between a single-threaded and multi-threaded execution of such a system can be observed when animating a group of characters; in relation to the population count and the character leg count?

1.5 Method Introduction

The aims and objectives of this thesis will be realized through an experiment. The experiment consists of several executions of the implemented locomotion algorithm, in which the execution time of the algorithm is measured. The experiment will increment the character count for each measurement. Measurements will be performed on both bipeds and quadrupeds, separately. Measurements are separately performed on both serial- and a parallel executions of the algorithm. The experiment is further described in Chapter 4. By doing an experiment which systematically measures and records the performance results, comparison graphs can be generated for the objectives.

Another possible method, which was not chosen for this thesis, is for example a user case study of visual quality of the generated result. Instead, an already visual quality tested algorithm (using closeness to motion capture data) was implemented (see Section 1.6) for focus on run-time performance evaluation.

1.6 Contribution

This thesis presents a novel standalone implementation of a pose- and feedback-driven locomotion system. It is based upon the previous work on locomotion algorithms by Coros et al. [8, 9] and Yin et al. [60]. It differs from previous algorithms in that it relies on BulletPhysics for solving the generated torques, it supports both serial- and parallel execution and it supports both bipeds and quadrupeds (with a flexible spine for quadrupeds) using the same algorithm. It furthermore presents performance measurements of the algorithm for varying amounts of characters, for both serial and parallel execution. Figure 1.5 presents a screenshot from the finished application and the resulting simulation of the locomotion algorithm implemented.

1.7 Thesis Outline

Following this introduction is Chapter 2, *Background and Related Work*, which provides an analysis of existing literature within the domains touched upon by



Figure 1.5: Screenshot of the application developed for the thesis. It shows walking bipeds and quadrupeds, simulated using the implemented pose-driven feedback locomotion algorithm. In this screenshot an example of interactivity is shown by having boxes thrown at the characters to disrupt their balance.

this work. It starts with an introduction on what have influenced the development of various procedural methods of animation, with focus on locomotion, as well as descriptions on terminology. The dynamics-driven procedural methods for locomotion mentioned in this chapter are described further. Chapter 2 also provides more insight and introduction to specific terms in the surrounding areas which contribute to the basis of this work, such as animation-, locomotion- and parallelism theory.

This is then followed by Chapter 3, *Implementation*, which introduces the model of choice for the procedural animation itself and its components, as well as supporting components and measuring methods in the finished application.

Chapter 4, *Method*, describes the method and settings for performing the experiment. Chapter 5, *Results*, then presents the following observed results. An analysis of the results, showing the rate of change when increasing the amount of characters, is presented in Chapter 6, *Analysis*.

Chapter 7, *Conclusions and Future work*, concludes the thesis and provides discussions on the experiment, the experiment results and the locomotion algorithm implementation.

Chapter 2

Background and Related Work

There is an absence of available research on performance of procedural animation of locomotion for serial and parallel multi-character simulations using dynamics-driven techniques. This is likely due to the fact that there is still a lot of ongoing research in finding more robust and visually pleasing simulations [13,44]. Pronost et al. briefly mention a $4 \times$ real-time performance for a, presumably serial, dynamics-driven animation system [44]. Coros et al. report a performance of $10 \times$ slower than real-time for 16 physically simulated characters [8]. Fang et al. reports real-time results [11] for their kinematic- and dynamics-driven animation system of myriapoda. Geijtenbeek et al. mentions their simulation having high performance [14] and being able to simulate in real-time. Johansen provided some real-time performance measurements and hardware specifications for his kinematic parametric locomotion system [21].

Modern interactive software and games are running in environments which supports multi-threaded applications. There is thus an opportunity to investigate performance variances and relations between parallel and non-parallel executions of an algorithm.

There exists a large amount of research on various forms of techniques for procedural animation of locomotion [8,9,13,19,21,25]. Especially the number of published research on physics-based procedural animation has been showing an increasing trend [13, p. 11].

All of this, as well as the pure aesthetic aspects that exist within games as a medium will impact the choice of algorithm model for procedural animation. In regards to developer usability, one should also take into consideration the harm that too much complexity or too little control might have on the development pipeline for artists and programmers.

This chapter begins with establishing a common understanding of domain specific terminology that will be used throughout this thesis. This is done by first providing a short introduction on the various aspects of animation quality and the existing ideas for this field and how they, in different ways, have influenced earlier works of procedural animation. Following this are explanations of terminology used to describe various aspects of locomotion which are commonly used within the fields of biology, robotics and animation.

After these introductory sections on terminology, an analysis of existing works on different procedures of procedural animation is presented, along with their reported properties in regards to result, implementation data and functionality. In the concluding part of this chapter a brief review of existing paradigms for parallel programming are presented as a foundation for the parallel execution of the implemented locomotion algorithm.

2.1 Locomotion

As mentioned in Chapter 1, locomotion is but one aspect of character articulation. It can be used to provide one part of the puzzle to construct believable virtual characters.

2.1.1 Believability in Animation

When discussing the quality aspect of animation of virtual characters, one need to make a distinction between believability and realism. In order for a person to perceive a virtual character as believable it does not necessarily need to be realistic [42]. Believability is sought after in interactive applications containing virtual worlds and characters in order to create an engaging experience for the user [41]. Believability is often achieved by a combination of several expressive behaviours and by being able to react to events and interactions accordingly [5]. Bates et al. conducted surveys using procedural animation of exaggerated emotions on cartoon characters in order to see on what level the users connected with the characters [5]. By combining several forms of character articulations one can thus increase the chances of creating believable virtual characters. Locomotion, being one form of articulation for movement is thus one important part in order to realize this. One example is the work done by Unuma et al. [54], in which various forms of gait were used in combination with full body poses to convey amount of exhaustion in a character.

2.1.2 Terrestrial Locomotion Terminology

A single gait cycle for a character contains several step cycles for each of the character's feet. A step cycle is described by its *duty factor* and *step trigger* parameters [1, 19]. The duty factor is the percentage of ground contact of the foot in regards to the whole cycle and the step trigger is the offset from which a foot's cycle starts. The time in which the leg is in the air is referred to in this thesis as the *swing phase*, while its time on the ground is referred to as the *stance phase*. Figure 2.1 shows a visualization of various gait cycles.

To differentiate between various movements within the reference frame of the character, they are categorized by the planes in which they exert movement.

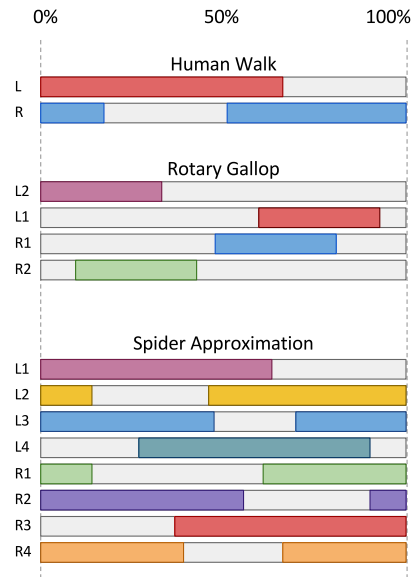


Figure 2.1: Gait cycles, defining the phase shifts and duty factors. Here presented in percentages of total cycle time. The left notations represents the left- and right legs, with numbering going from front- to back. The stance phases are represented as the colored bars, while the swing phases are shown as light-grey bars. The graphs are adapted from previously published gait data [4, 6, 56].

The *sagittal plane* is the plane which vertically spans from the rear to the front, effectively creating the character’s left- and right hand sides. The *coronal plane* is the vertical plane which divides between the front and back of the character. Lastly, the *transverse plane* is the horizontal plane which defines the top and bottom of the character. These planes are shown in Figure 2.2.

2.2 Dynamics-Driven Procedural Locomotion

Dynamics driven procedural animation uses a physics solver of some kind to generate the final articulated angles and positions for the limbs of a character. The means of which the torques and forces are calculated for the solver to apply may vary. Where kinematic solutions use inverse kinematics to find the angles for intermediate joints for the extremities, the dynamics-based equivalent is an inverse dynamics calculation to find the correct torques and forces to perform a wanted motion for the extremities to reach their end targets.

Full inverse dynamics are exceedingly computation heavy. They are not, as of yet, a realistic approach for real-time applications [44, 58]. Instead of full inverse dynamics, various forms of approximations are used, of which each are specialized for a certain type of articulation. Such specialized approximated methods will

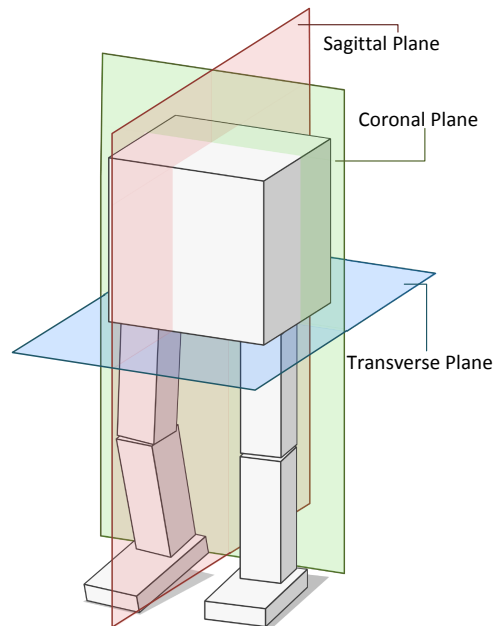


Figure 2.2: *The planes used as reference frames to express movement in the local space of a character.*

also commonly rely on character specific input parameters. In order to find these character specific parameters there is often a need for some sort of *optimization* [1, pp. 10-12] phase to find a set of parameters that result in a robust gait [9,14,44,57], though some models manages to avoid it [8,25].

Gait optimization phases are used to *teach* a character with an animation model to walk. This is done by ever so slightly change a set of input parameters for the model and the character, and test the result for each change [9]. One method of finding an optima of parameters is by letting the result be graded by a set of wanted criteria, and perform some form of *greedy* selection [31]. Karl Sims [50] introduced the concept of virtual critters that, through simulation of evolution, could learn how to propel themselves in their virtual environments. His model simulated several generations of critters. In this model, the most successful critters where allowed to evolve and change behaviour and shape.

For procedurally animating the locomotion of a character for a physics solver, there exist three major types of methods, with focus on real-time animation for games and interactive simulations, for which the majority of academic research has been made. The following sections (Section 2.2.1, 2.2.2 and 2.2.3) outline these and how they differ from- and relate to each other.

2.2.1 Inverted Pendulum Methods

The *Inverted Pendulum* (IP) model [24, 28, 45, 53] has been used as a way for finding foot placement for balancing dynamics-driven characters. The swinging motion apparent in the movement of the legs in bipeds have shown to be achievable by approximating the legs with an inverted pendulum [29]. The inverted pendulum model does this approximation by neglecting the knees. Full leg movement can then for example be provided kinematically upon the solved foot placements by the use of inverse kinematics.

The balancing of quadruped characters can also be approximated by the IP-model, this by utilising two pendulums; one for each pair of legs [1,15]. Kenwright builds upon the model for bipeds to make it able to generate more aesthetically pleasing result [25]. This by for example introducing the use of an elongated body representation instead of a single point as well as impulses, to better visualize walking on uneven terrain and to get more dynamic results.

The model for humanoid bipeds can also be expanded by using a spring-damper setup for each leg, as these act as an acceptable approximation of the human muscles [23]. The inverse pendulum model can be used as an external foot placement model for other dynamics-driven locomotion solutions [8].

2.2.2 Muscle-Driven Methods

Dynamics-driven methods may generate motion parameters by simulation of muscles. Though it generally adds additional layers of complexity compared to other methods, it is regarded to achieve the most realistic results. As the strength and energy expenditure of these virtual muscles may be altered, they prove useful in interactive medical simulations [57].

Wang et al. presented a solution for procedural locomotion using a Musculoskeletal driven dynamics model [57]. It used a virtual *Hill-type* muscle model [14, 57] to apply biologically constrained forces upon the limbs. An off-line optimization phase using biologically based fitness functions, such as the amount of effort needed by the model [57], was used to fine-tune the gait parameters driving the model in order to achieve realistic movement and allow for simulation of various muscle deficiencies.

Geijtenbeek et al. improved on this method by introducing an off-line optimization phase that routed muscles along the skeletal hierarchy automatically [14]. This method lessened the need for biologically correct muscle data during construction of the character model. It further allowed for the creation of virtual creatures with fitting gaits based on their anatomy. Creatures with more unrealistic proportions thus evolved gaits to compensate.

2.2.3 Pose-Driven Feedback Methods

Dynamics-driven movement can also be generated from a set of kinematic targets [13]. Motion graphs of kinematic animation data contain the wanted trajectories that the physics model should follow as best as it possibly can. This model further relies on concepts common in the fields of robot- and control logic to allow for the characters to achieve the predefined movement from the graphs as close as possible. Feedback *Proportionate Derivative* (PD) controllers are used to generate variable torques on each joint in order to match the motion graphs.

It is worth noting that in PD-controllers an additional integral term can be implemented as well to eliminate steady state errors (difference in the set goal and the actual steady state of the controller). This type of controller is then known as a PID-controller. This is however of limited use for character animation [13, p. 32], where the goal is often constantly changed (as it is being animated).

Raibert et al. developed early robotics inspired models for dynamics-driven biped- and quadruped gaits [45]. *SIMBICON* [60] by Yin et al. built upon several of these early concepts to produce stable biped gaits that could learn from a set of artist created key frames. This model was optimized to withstand pushes, without falling, to some degree. It has since then been further expanded on-, or inspired methods similar to it to support a wider arrange of biped actions [36] and quadruped gaits such as galloping [9]. The use of *virtual forces*, another concept from the field of robotics [40, 43, 52], have been used in some models [8, 9] in order to compensate for gravity and allow for lower PD-gains, in turn allowing for less stiff movements.

The motion graphs used for animation can be created by an animator (as key-frame data) or by using mathematical functions. Motion graphs can also be extracted from motion capture recordings. This motion graph data, along with the control system parameters and PD-gains are then generally subjected to some form of optimization as well.

Compared to the musculoskeletal driven models, the graph driven models might result in less realistic movements. However, as it conforms to the wanted movement solution for a character rather than finding the optimal movement solution based on anatomy, it might leave more room for artistic freedom and thus be more usable in games.

The optimizations needed for many dynamics-driven methods may use motion capture data or rules on acceleration, velocity and amount of rotation. Several iterations are executed, each with a new perturbation on dependent parameters. Fitness for a set of parameters in an iteration is evaluated to allow the model to find its optimal parameters for how to move and balance [9, 14, 36, 57]. Some local optimization techniques that have been used for locomotion controllers includes for example *stochastic greedy algorithms* [9, 31] and *covariance matrix adaptation* [36, 44].

The implemented algorithm in this thesis (which will be measured in the

experiment) was chosen to be a graph-, virtual force- and PD driven model (effectively a pose-driven feedback model) as described by Coros et al. [9]. Some influences were also taken from two similar solutions presented by Coros et al. [8] and Yin et al. [60]. The implementation done for this thesis is further described in Chapter 3.

2.3 Parallel Execution of Locomotion Algorithm

There is a possibility that character animation in a game will be executed for several characters. As a serial implementation of an algorithm grows in size, it might become relevant to investigate whether it lends itself to parallelization or not.

The animation methods presented in Section 2.2 all contain somewhat exploitable phases of various levels of *data parallelism*. Data parallelism is possible when each processing unit in a system can execute the same instruction set on several data sets.

Data parallelism contrasts to *task parallelism* which might execute differing instruction sets over threads. Data parallel components are well suited for parallel execution due to their inherent independence from component-to-component communication (which might require synchronization). A data parallel approach to algorithms for execution are one important quirk [12] to SIMD-like (Single Instruction Multiple Data) architectures as they perform poorly when experiencing too many branching instructions.

Even though the architecture of modern CPUs are better suited for code with branching patterns compared to SIMD architectures, they also do have the possibility to benefit from data parallel algorithms [32].

2.4 Performance in Parallel Algorithms

Amdahl's law [2] and *Gustafson's law* [16] tell what changes in performance can be expected when executing an algorithm in parallel. Both of these laws are theoretical in nature and does not consider the multitude of bottlenecks that occurs when implementing a parallel algorithm. They do however pose two interesting outset views of how one can approach parallelization. Amdahl's law states that the problem size of an algorithm is constant, and shows how the effect of a parallelization of an algorithm with fractional possibility of parallelization, results in a steadily declining performance gain as one increases the computation capability.

Gustafson's law, on the other hand, disputes this as the only scenario and claims that in some cases it is more likely that a large problem set is to be solved within a certain *time limit*, and thus, the problem size is increased along with the computational power, while the time limit remains constant.

An algorithm of data-parallel nature may be parallelized on the CPU using classic parallel transform features of libraries such as *OpenMP*, *Parallel Patterns Library* or *Intel Threading Building Blocks*. These methods are simple to implement for an existing single threaded algorithm, provided all instances of it can be executed at once, without the need to wait for each other.

The parallelization of the locomotion algorithm will be done on the per-character level, in order to keep the design of the single- and multi-threaded variant of the algorithm as close to each other as possible. The number of per-character algorithm executions to be done in serial per parallel invocation is described in Chapter 4. Further described in Chapter 4 is the choice of time limit to be used for the experiment.

2.5 Summary

The first part of this chapter provided a review on existing research on the subject of procedural animation of character locomotion following. Also presented was a brief overview on various quality aspects of animation with respect to believability for richer user experiences in games and interactive software. It also brought forth the various sub categories of procedural animation as well as a closer look on the existing methods of realising locomotion procedurally. Following this was a brief overview of the various aspects of parallelism.

In Chapter 3 a motivation for the choice of locomotion algorithm is presented. This is followed by descriptions of the various phases and components present in the chosen algorithm. It also presents the chosen solutions for parallelism of the algorithm as well as descriptions of the overlying framework and the methods of performance measurement.

Chapter 3

Implementation

In Chapter 2 locomotion terminology was introduced for virtual simulations of terrestrial locomotion.

As the experiments outlined in Chapter 4 only aim to evaluate performance relations for an animation model and not its resulting visual quality, consideration needs to be taken for the choice of animation model with regards to its reported findings on visual quality.

Other aspects of importance for choice of model, that are also somewhat based on existing work, are the level of complexity introduced to the user (or developer) and the availability of control presented. Worth noting as well is that the domain of games might not always favour realism, but rather believability in a certain setting, as well as artistic freedom.

This chapter begins with a summary and description of the chosen model for procedural animation of locomotion, along with a motivation with consideration to the domain. This is followed by separate descriptions of the various components and phases present in the model, as well as their interactions. The chapter is concluded with an overview of the interactions of the stages of the model.

3.1 Implemented Algorithm

The implementation algorithm chosen for the procedural generation of terrestrial locomotion, is a dynamics-driven model.

As mentioned in Chapter 2, dynamics-driven locomotion calculates the torques and forces needed for a physics solver to generate the final character motions. A big aspect of why a dynamics-driven model is favoured instead of a kinematics-driven model, is due to a physics engine's tendency to convey more physically realistic movements [13,22], although not necessarily biomechanically accurate [13]. The dynamics driven model chosen is a pose-driven feedback model and has its roots in the SIMBICON algorithm for dynamics-driven locomotion of bipedal characters.

Specifically, the implementation outlined in this chapter is based on the concepts presented by Coros et al. [8,9] and Yin et al. [60]. It is not built upon the code bases of these former works, it is a standalone implementation. The

implemented model differs from existing methods in the way that it as a single model supports both bipeds and quadrupeds.

The main aspects of driving the internal articulation of a leg using PD-controllers and virtual forces is carried over to this solution, as is all the steps regarding the animation of the leg and leg frames. Animation for head, neck and tail has not been implemented for this thesis. A novel aspect of the implementation for this thesis is the execution of locomotion for both bipedal and quadrupedal characters using the same algorithm, for both single- and multi threaded execution. A data driven paradigm is used where data is separated from logic to allow for simple data parallel [10] execution.

A pose driven feedback dynamic method was decided upon as it promises some artistic freedom as the motions can be somewhat controlled by the motion graphs (the motion trajectory over time) for which the legs and feet tries to match their cyclical movements [8,44,60]. This kind of method also integrates well with common physics engines for games, with the promise of a lower performance impact [44]. It also promises a possibility for a larger amount of possible gaits compared to the basic implementations of inverted pendulum models [60].

One drawback when relying on PD controllers for joint control is the added complexity of manual tuning of gains required [53], though this can be alleviated somewhat by automatic scaling based on proportions [8]. For this implementation, the gains were manually tuned.

Muscle-based dynamics simulation has proven to more effectively generate realistic movement [14], but it was not chosen as it adds an extra layer of complexity for creating- and especially for optimizing characters for simulation.

Multi-threading was implemented using the *parallel-for* paradigm for non-intrusive implementation of data parallel execution. Both the Parallel Patterns Library by *Microsoft* as well as OpenMP was implemented for this application. For the experiment OpenMP was used, being the more cross-platform alternative.

The full articulated armature that is handled by the physics engine can be seen in Figure 3.1. The *overall control loop* [9], or *controller*; acts on a simplified abstraction of the armature, see Figure 3.2. This makes the controller somewhat independent from the anatomy of the skeleton [9], such as the amount of joints in a leg. Thus the same controller logic can be used for both bipeds and quadrupeds, of various sizes.

A pair of legs of a character are attached to a *leg frame*, which acts as the local coordinate system for that pair of legs [9]. A quadruped has two leg frames, one front leg frame and one back leg frame, while bipeds have one. The leg frames of a quadruped are connected by a chain of joints acting as the spine. The leg frame tries to follow a predefined rotational trajectory by providing a feedback of the torque back through the legs in stance.

The physics solver will then interpret the skeleton as a series of connected joints with various torques to be applied. Each leg will apply different torques

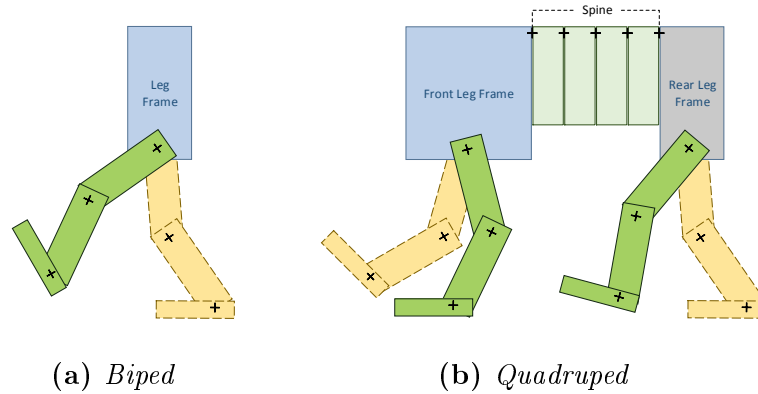


Figure 3.1: Controller layout as a collection of rigid bodies with joints as seen by the physics engine. The leg frame is here treated as a single rigid body in the biped (a), while the quadruped has two connected leg frame rigid bodies (b). The quadruped uses a spine made up of an array of connected rigid bodies with joints to connect the both leg frames.

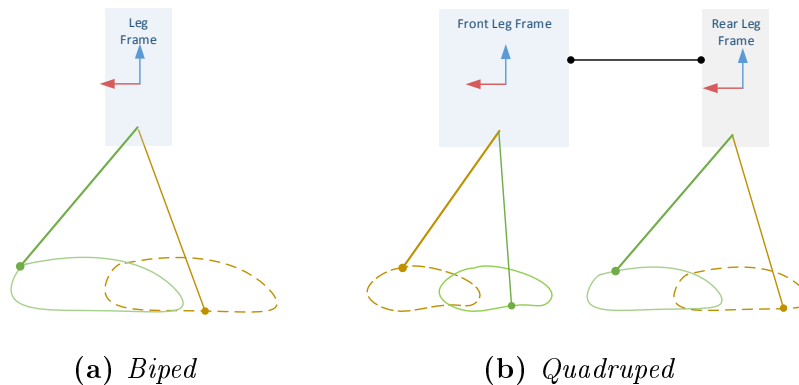


Figure 3.2: Controller abstractions for generating propelling motion. These simplified models are what the controller uses to generate the overall motion instructions, which are then handled in finer detail by various sub systems. The leg frames acts as the local reference frame for the simplified legs and aims to follow a predefined trajectory. The biped (a) make use of a single leg frame, while the quadruped (b) has one for the front- as well as the rear. Note that at this stage, the number of joints are not taken into consideration.

depending if it is in a stance- or swing phase in its animation. A stance leg tries to accomplish the torque needed to keep the leg frame stable. The angle of the knees are calculated using inverse kinematics that defines a goal angle for which a PD driver estimates the torque. The feet tries to reach predefined lift- and strike angles.

Following are descriptions for the various stages of the model and their importance. The interaction between these stages are then presented in Section 3.4.

3.1.1 Gait Player

The gait player can be seen as the overlying animation player, which increment the normalized gait phase $\Phi = t/T$, where t is the current gait time and T is the total gait period. The progress, ϕ , in swing and step phase for the legs are defined by the step triggers and duty factors of the legs, as well as early or late foot strikes (when the physics engine registers a foot strike before or after the model determined it to happen). An early foot strike will immediately force the current phase from swing to stance if $\phi > 0.8$ in the swing phase. A late foot strike triggers a temporary downwards offset to the foot target position.

3.1.2 Virtual Forces

In order for the abstracted model to be viable, the model takes use of virtual forces [9]. Virtual forces are a form of virtual control method which serves as a means to be able to control general motions in detail while at the same time be able to exert corrective torques and forces to allow for these motions in an unknown environment [43, 52].

A virtual force is expressed as the force we want a certain limb or leg frame to try to achieve, even though this limb might be more complex than the leg vector for which the motion is controlled. This force is achieved by translating the force vector to a set of torques. Figure 3.3 shows the resulting torques for the chain of joints in the affected limb. Following is the method for calculating torque from a virtual force applied to an end effector, as described by Coros et al. [9] and Geijtenbeek [13].

In order to make the full limb exert the wanted virtual force, the Jacobian transpose is used for each degree of freedom (DOF) in the limb in order to approximate the sets of torques, τ_F , for each joint which will amount to the wanted force as applied on the whole limb (3.1).

$$\tau_F = J(p)^T F \quad (3.1)$$

The Jacobian Transpose matrix (3.2) is composed of rows, each defining the rate of change for a point p around a chain link's DOF, α_i as expressed as the normalized axis of rotation in world frame coordinates [13, 44]. The number of

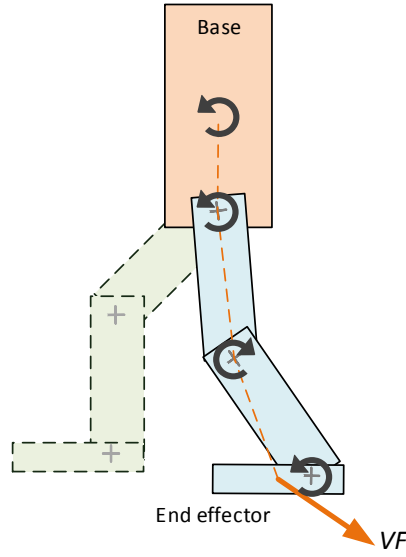


Figure 3.3: A virtual force applied to an end effector resulting in a set of torques for the joints in the limb.

rows, k , equal the total DOF count in a chain. Thus, for example; a 3-DOF joint equals three rows in the matrix (3.2).

$$J(p)^T = \begin{bmatrix} \frac{\partial p_x}{\partial \alpha_1} & \frac{\partial p_y}{\partial \alpha_1} & \frac{\partial p_z}{\partial \alpha_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial p_x}{\partial \alpha_k} & \frac{\partial p_y}{\partial \alpha_k} & \frac{\partial p_z}{\partial \alpha_k} \end{bmatrix} \quad (3.2)$$

For each DOF i in a chain, the virtual force F is thus transformed resulting in the torque exerted by that joint for that force (3.3). Note that this is only a part of the final torque. All torques generated by the control loop are summed before stepping the physics solver.

$$\tau_{F,i} = (\alpha_i \times (p - b_i))^T F \quad (3.3)$$

3.1.3 Leg Frame and Torque Feedback

The leg frame exerts the torque $\tau_{leg \ frame}$ (3.4). The desired torque for a leg frame, $\tau_{d_{leg \ frame}}$, is calculated using a PD-controller working towards an orientation as defined by a motion graph.

$$\tau_{leg \ frame} = \tau_{stancelegs} + \tau_{swing \ legs} + \tau_{spine} \quad (3.4)$$

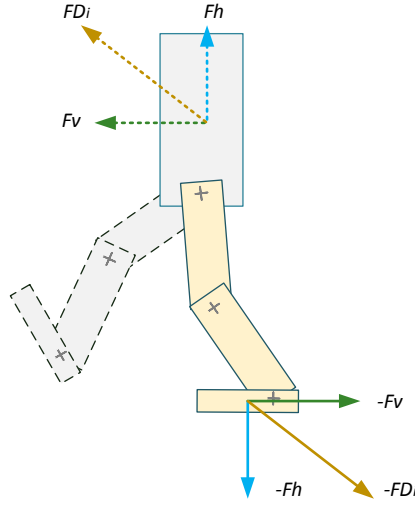


Figure 3.4: The virtual forces acting on a leg frame through a stance leg. The opposite of the forces are applied on the stance foot with the leg frame as base point.

The leg frame then works to achieve this torque $\tau_{d_{leg\ frame}}$ by feedback to the stance legs (3.5). The torque $\tau_{stance\ legs}$ is split evenly among the legs in stance. It is the finalized torque for the stance legs, which is supplied to the physics solver.

$$\tau_{stance\ legs} = \tau_{d_{leg\ frame}} - \tau_{swing\ legs} - \tau_{spine} \quad (3.5)$$

3.1.4 Legs

Virtual forces act on the leg frames through the legs which are in stance. The calculation of virtual forces assumes a stable base point in order to give good approximations. Coros et al. argue that the proportionally larger mass of the leg frame makes it more viable as a base point rather than the feet [9]. The sign of the summed leg frame force is flipped and this resulting opposite force is then applied on the stance feet as end point, with the leg frame as base. Following are the forces exerted upon the stance legs and their leg frame.

A force F_h is applied to the feet to regulate the height of the leg frame based on a PD controller working from a motion graph. A virtual force F_v (3.6) on the feet is used to regulate the velocity of the leg frame, where K_v is an optimized gain factor, v_d is the desired velocity and v is the current velocity. The velocity parameters of F_v are further discussed in Section 3.1.6.

$$F_v = K_v * (v_d - v) \quad (3.6)$$

A phase dependent force $F_D(D)$ is applied to each stance leg, where D is the amount of forward internal progress of the foot with the leg frame as reference frame (3.7). Figure 3.4 showcases the forces that acts upon a leg frame through a stance leg.

$$D = |\vec{P}_{leg\ frame} - \vec{P}_{foot}| \text{ (projected to ground plane)} \quad (3.7)$$

The constant parameters of $F_D(D)$ (3.8) are defined through optimization. This force allows for fine detail tweaking of the motion of each leg [9].

$$\begin{aligned} F_D(D)_{vertical} &= c_0 + c_1 * D \\ F_D(D)_{horizontal} &= c_2 + c_3 * D \end{aligned} \quad (3.8)$$

These forces mentioned are not in effect on the leg frame if there are no legs in stance.

For the swing legs, a virtual force F_{sw} is applied to allow for lower PD-gains. Lower PD-gains result in less stiff movements. F_{sw} uses a PD controller with a variable proportional gain $k_{ft}(\phi)$ based on an optimized graph.

This PD-controller drives the foot from its current position to its strike position. The derivative gain is set to ten percent of the proportional gain. Note that this solution for F_{sw} differs from the use of a spring-damper solution by Coros et al. [9].

The virtual forces does not guarantee correct placement of the knees. To alleviate this a two-link inverse kinematics solution, using the law of cosines, is used to calculate the correct knee position (based on the hip- and foot positions). PD-controllers in the joints are then used to calculate additional torque for the leg joints in order to reach the wanted internal joint angles.

3.1.5 Gravity Compensation

To allow the model to compute motions that neglect the effects of gravity (but still allowing gravity, for realism and for the character to stay grounded), a set of gravity compensation virtual forces are applied to the joints. For each affected leg joint, a force $F_g = -mg$ is applied to its center of gravity. Gravity compensation is performed on the swing legs and the spine. The spine segments effectively get two virtual forces each, one for each leg frame as base. The virtual force of the spine segments are weighted linearly based on their distance to the base link. Each spine link thus receives a virtual force (for each leg frame as base) of $F_g = -mg * w$, where $w = 1 - d$ for the normalized distance $0 < d < 1$ to the base leg frame. Figure 3.5 shows all the gravity compensation forces distributed over a quadruped character with two legs in swing phase.

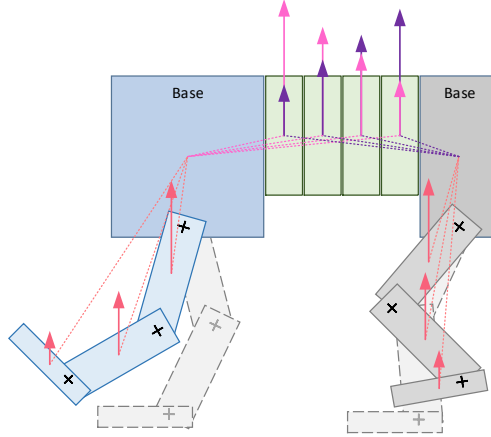


Figure 3.5: *The composition of gravity compensation forces applied to a quadruped character. The forces are applied to swing-phase legs and spine links. Note that each of these forces will result in fractional torques for the preceding joints of the end effector where the force was applied.*

3.1.6 Foot Placement

Desirable foot placement, for which the controller strives to achieve is velocity based. From the foot's current placement P_0 a new velocity scaled target position P_1 is calculated as $P_1 = P_{leg\ frame} + (v - v_d)s$. Where $P_{leg\ frame}$ is the new, non-scaled, world space foot position as defined by a motion graph and offset by the leg frame's current position. This position is then further offset by the scaled difference between the current velocity v and the desired velocity v_d .

The trajectory for the foot between P_0 and P_1 follows an optimized step height path $h_{sw}(\phi)$ in the sagittal plane, where ϕ is the current progress of the phase. An optimized easing graph $t_{sw}(\phi)$ furthermore enables variable horizontal movement speed for the foot.

The desired velocity v_d is a value that incrementally tries to reach the user defined goal gait velocity. It is the current velocity, incremented or decremented by a maximum value per second (each frame) until it reaches the goal velocity.

The foot angle in the sagittal plane, is based on the transitions between the swing- and stance phases of the leg. The foot is driven by a PD controller to the predefined and optimized take-off- and strike angles that occur in the transitions. Time offset parameters for when to switch between target angles are tweaked in the optimization phase as well.

3.1.7 Proportional-Derivative Controllers

The internal shape of the legs, as well as the foot angle and the spine shape are controlled using PD-controllers. A PD-controller is a feedback reliant means to minimize a general error value. This error can for example be a difference between a wanted and a current angle of a joint. A partial torque for a joint in the character armature can thus be estimated using a PD controller (3.9).

$$f(t) = K_p e(t) + K_d \frac{d}{dt} e(t) \quad \text{where } t=\text{time and } e=\text{error} \quad (3.9)$$

The gains K_P and K_D are application dependent as is the interpretation of the result value $f(t)$. For joint torque control, the result value is directly applied as partial torque.

3.2 Constrained Rigid Bodies using BulletPhysics

The physics library *Bullet Physics*, by Erwin Coumans, was used for the physics simulation. It was chosen as it is an open source engine with constraint support. It supports 6-DOF generic constraints which is useful when having constraints with variable DOFs. Bullet Physics is used in a variety of interactive applications and games. Parts of Bullet Physics was developed in co-operation with video game developer and publisher *Rockstar Games* and integrated in their game engine *RAGE*, which in turn powers *Grand Theft Auto IV*, *Grand Theft Auto V* and *Red Dead Redemption*¹, amongst others. Bullet Physics has also been used in various movies such as *Shrek 4*, *How To Train your Dragon* and *Megamind*². Bullet Physics has been deemed to be one of the best performing open source alternatives for physics engines [7].

For all simulations in this implementation, Bullet Physics was executed at a rate of 120Hz.

3.3 Optimization

An off-line optimization phase was run for both the biped and quadruped character to tweak the default parameters (presented in Section 3.1) defining the various run-time timings and motion graphs. The optimization was done using a greedy stochastic local optimization method, as presented by Coros et al. [9]. A greedy stochastic algorithm extends a general greedy algorithm by introducing random perturbations [31].

The optimization only needs to be done once per character. Once the parameters are found, they are saved to disk and used for all other executions of

¹www.bulletphysics.org/Bullet/phpBB3/viewtopic.php?p=11971, Accessed 2014-11-09

²www.bulletphysics.org/wordpress/?p=241, Accessed 2014-11-09

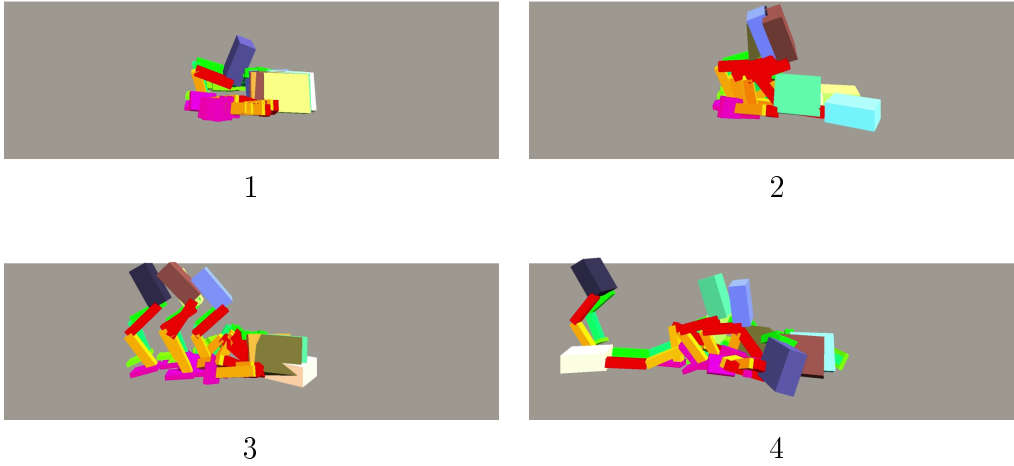


Figure 3.6: *Learning to walk. The images, appearing in chronological order, are showing iteration stages in the optimization phase. A number of variations of the previous winning parameter set is tested each iteration and evaluated.*

that character. The characters that were simulated in the experiment used such optimized parameters to ensure that they walked. An example of the resulting improvement from optimization of the walking gait for the biped can be seen in Figure 3.6.

The parameter tweaked per controller was the gait period time. The parameters tweaked per leg frame were: desired orientation graph, desired leg frame height graph, foot tracking gain graph, desired step height graph h_{sw} , foot swing transition easing graph t_{sw} , F_h PD-gains, F_v scaling-gain K_v , the constant parameters of $F_D(D)$, the foot take-off angle, the foot strike angle, the scaling factor s for foot placement and the step lengths in the transverse plane.

Each graph was represented as a piecewise linear function with 4 single precision float data points each (linearly interpolated when read).

The parameters tweaked per leg were: the step trigger, the duty factor, the foot take-off time and the foot strike time.

$$f_{obj}(P) = w_d f_d + w_v f_v + w_h f_a + w_r f_r \quad (3.10)$$

A greedy stochastic algorithm aims to minimize the result of a function by changing the parameters set to be tweaked. The function to be minimized (3.10), f_{obj} is composited of several weighted measurements on the character gait performance. Based on these rules a gait resulting from a certain set of perturbed parameters is graded. This evaluation is repeated several times, and each time a new best grade is recorded, the old parameter set is discarded and the following perturbation is based on those of the current best.

The component f_d measures motion deviation. Here Coros et al. used a

motion recording of a real dog during various gaits. As motion capture data was not available as a part of the implementation for this thesis, a combination of desired character travel distance and leg IK angles and positions were used as desired motion references instead. This was deemed sufficiently good (as the main objective of this thesis is performance measurement) to achieve a character able of forwards propulsion, though one might discuss the quality aspects of naturalness and realism of the end result.

The other components: f_v , f_a and f_r ; respectively measures velocity deviation, leg frame acceleration and leg frame rotation deviation.

When perturbing a parameter set, there is a 20% chance for a parameter to be changed. When changed, the parameter P is incremented by a uniform distributed random value ΔP . This random value ΔP will be between $P - 0.1R$ and $P + 0.1R$, where R is the difference between the allowed maximum- and minimum values for P .

Each random generator is initialized at application start with a seed value which is constant between application runs for a deterministic optimization phase.

3.4 Controller Logic

The overlying control loop relies on two main types of input: the character parameterization of animation aspect data (containing PD gains, virtual force DOF chains, as well as motion- and gait graphs) and the joint transform data of the character armature. The schematic in Figure 3.7 provides a simplified view of the dependencies between the main steps.

The main control loop, as described in Algorithm 1 then calculates the partial torques from the PD controllers and applied virtual forces which are then summed before being used by the physics solver. The result from the physics solver is applied to the armature and thus used in the following iteration.

The implementation uses the *Artemis* entity system framework by *Gamadu*. The paradigm of a data driven *entity-component-system* based software architecture is to separate data from logic. Each joint is represented as an entity in this architecture, and is composed of components defining data for rigid body properties as well as components defining constraint properties.

The character itself is also represented as an entity with a controller component, which effectively is the full parameterization of the character. Parameter perturbation during the optimization phase thus simply rewrites the data of this component.

Each component type in *Artemis* is processed by one or more *systems*. In the entity system paradigm, all major logic resides in the systems, and the systems each handle a specific task. In this implementation, the controller loop resides in one system, while handling and updating of rendering, physics and input are handled by separate systems.

Algorithm 1: Controller Update

Data: *controllers*[]=list of all controllers, *dt*=frame time

Result: τ []=array of all torques for all joints for all controllers

```

1 foreach  $c \in \text{controllers}$  do
2    $\Phi = \text{getPhase}(c)$ ;
3    $\Phi += dt/T$ ;
4   if  $\Phi > 1$  then
5      $\Phi - = 1$ 
6   end
7   // All actions on torque array  $\tau$  done
8   // only for assigned range of controller
9    $\text{clearMyRange}(\tau)$ ;
10   $\text{calcDesiredVelocity}(c)$ ;
11   $\text{updateFootPlacement}(c, \Phi)$ ;
12  if  $\text{hasSpine}(c)$  then
13     $\tau += \text{calcSpineTorque}(c)$ ;
14  end
15   $\tau += \text{calcVFTorques}(c, \Phi)$ ;
16   $\tau += \text{calcPDTorques}(c, \Phi)$ ;
17  foreach  $lf \in \text{getLegFrames}(c)$  do
18    // Calculate LF torque and correct
19    // the stance leg torques based on it:
20     $\tau = \text{calcLegFrameTorque}(lf, \tau, \Phi)$ ;
21  end
22 end

```

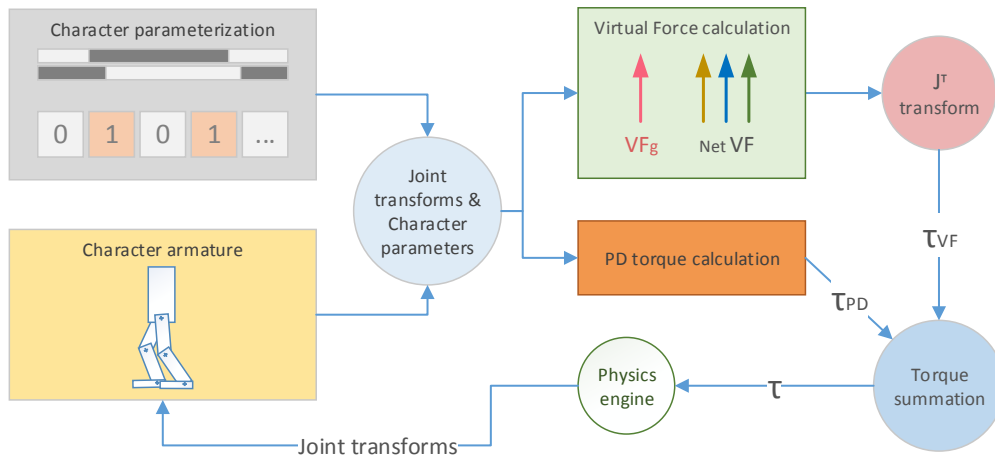


Figure 3.7: Overview of the overall control loop. The character locomotion parameters (including gait- and motion graphs) is used together with joint rotation and translation data to calculate the new torques. The new torques then result in updated joint transforms through the physics simulation. Some of the character parameters accessed will vary as well, based on the current gait phase Φ .

3.5 Summary

Described in this chapter were the various implementations done for this project. It introduced the algorithm of choice for procedural animation followed by descriptions of how this algorithm and its main components was implemented. In Figure 3.8 and Figure 3.9 the resulting locomotions are shown for the biped and the quadruped respectively. In Chapter 4, the experiment is described in detail, providing descriptions of how it was set up and executed.

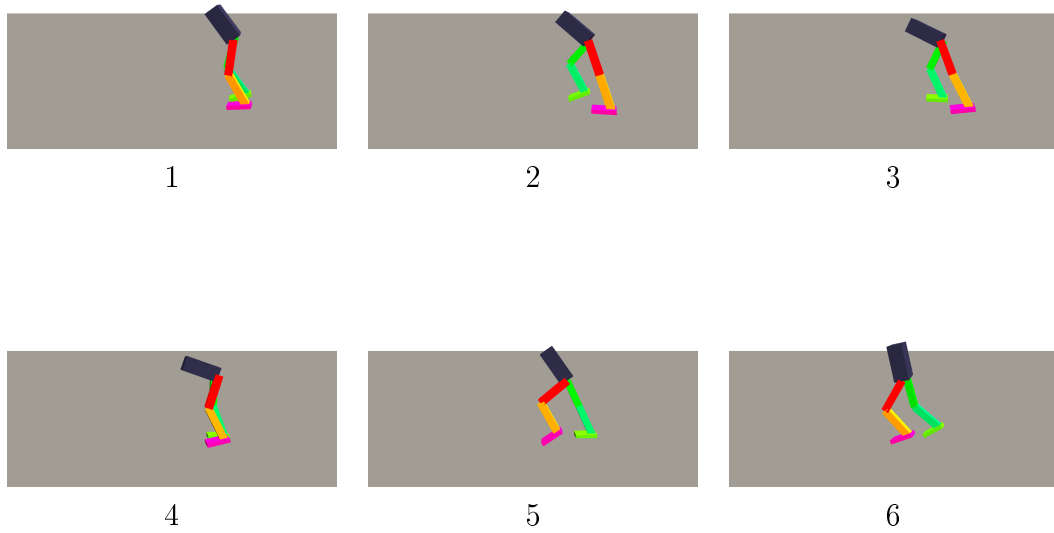


Figure 3.8: *Frames of the simulated biped locomotion. Appearing in order.*

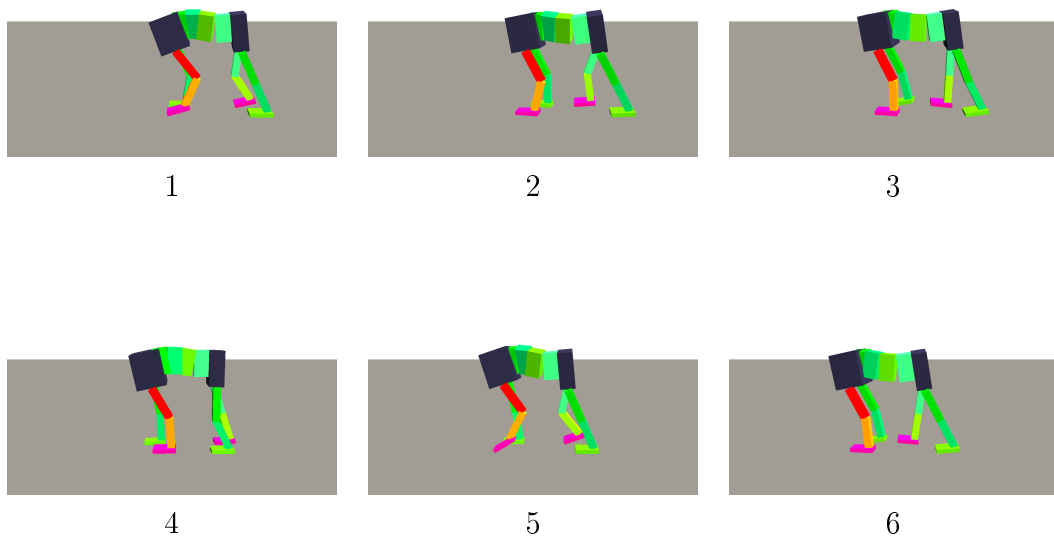


Figure 3.9: *Frames of the simulated quadruped locomotion. Appearing in order.*

The method used by this thesis is in the form of an experiment. In this chapter, the experiment used to measure the performance of the locomotion algorithm is described. The performance impact in the form of *milliseconds for locomotion algorithm execution time* as an effect of *character count* and *per-character leg count* is measured. Two scenarios, one serial and one parallel, defining the values for these parameters are set up. For each scenario, a scene of characters, with either two or four legs, are procedurally animated and rendered.

The measurements are of the locomotion algorithm part of the program only, and is not to be confused with the performance for the whole program or the execution of the physics solver [3].

4.1 Scenarios and Parameters

The experiment consist of two scenarios. The first is exclusively serial, while the second consist of increasing parallel loop invocations. Each scenario is a composition several smaller tests that changes parameter variables. The serial scenario (Figure 4.1) consists of the following variable parameters:

- The *character count*, which will range from 1 to 100, controlling the amount of characters in one run.
- The *leg count*, which will either be 2 or 4, representing biped or quadruped characters, respectively.

For each character count iteration a fixed number of algorithm *runs* will be executed for both bipeds and quadrupeds. Each run consists of 800 simulation frames (or steps). Figure 4.2 contains a chart of the composition of a single program execution in a scenario. The number of runs is a constant parameter value over all executions. A total of 20 runs are performed for each end measurement to calculate a mean and a standard deviation (to show the variation from the mean). Mean and standard deviation are calculated on a per-frame- and per-execution basis. All quadrupeds are simulated with 4 links in their spine.

Serial Scenario

Settings			Measurements	
Character count	Leg count		Milliseconds (avg)	Standard deviation
1..100	2	➔	<i>ms</i>	σ
	4	x Runs ➔	<i>ms</i>	σ

Figure 4.1: The setup and measurements of the serial execution scenario. An average measurement and a standard deviation is acquired for each character count specification, for either 2- or 4-legged characters.

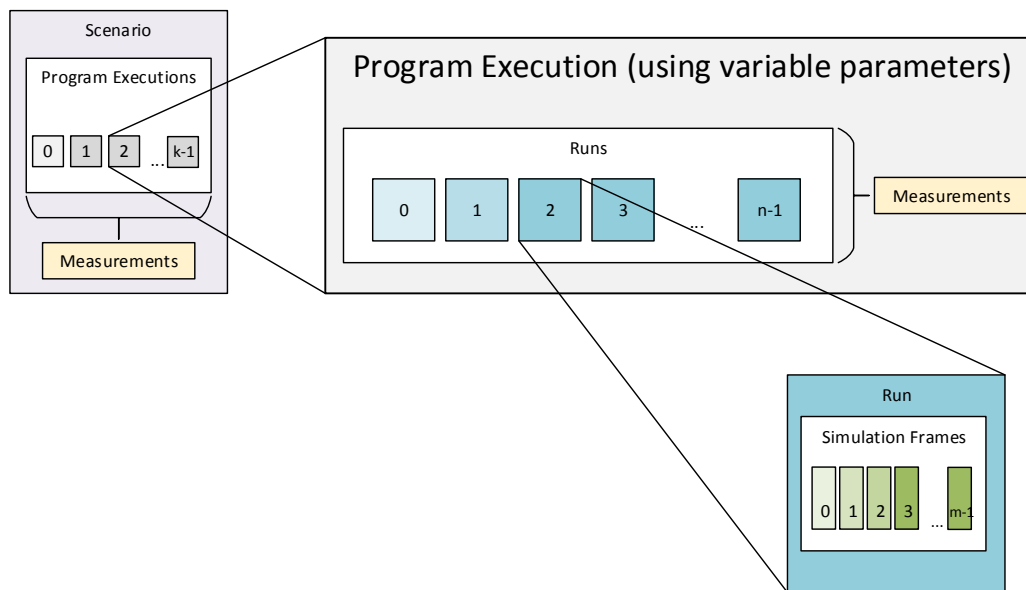


Figure 4.2: Relationship between scenarios, program executions, runs and simulation frames. The two scenarios are each composed of several program executions, one for each incrementation of the character count parameter. Each program execution will run several simulation runs to get a mean and a standard deviation. Each simulation run consists of a fixed amount of simulation steps (800), enough to simulate about eight foot strikes for the biped character and six strikes (per leg frame) for the quadruped.

Parallel Scenario

Settings				Measurements	
Parallel loop invocations	Character count	Leg count		Milliseconds (avg)	Standard deviation
2	1..100 / 2	2	x Runs	ms	σ
		4		ms	σ
3	1..100 / 3	2		ms	σ
		4		ms	σ
4	1..100 / 4	2		ms	σ
		4		ms	σ

Figure 4.3: The setup and measurements of the parallel execution scenario. The amount of characters per parallel invocation is the quotient of the total character count and the number of parallel invocations.

The character count value is based around the number of character simulations that are possible to execute serially within a set time limit. The time limit is based on the amount of time available per rendered frame of a real-time interactive application. A real-time application, such as a game, should be able to output between 30 and 60 frames per second in order to convey believable fluidity to the user [38, 39].

In an application that needs to output 60 frames per second, the total available time budget per frame for the entire application is: $b = 1/60 \approx 16.67ms$. As the algorithm in this thesis in itself was less performance intensive per simulation step than the surrounding systems, it would not exceed this time budget before indirectly causing dependent systems to do so. For the biped, the total frame time would exceed the time budget around 200 characters. For the quadruped it exceeded the time budget around 100 characters. The lowest of these values, 100, was chosen as the value for maximum character count, for both.

The performance of the algorithm for each simulated frame in each run is logged and stored. The measurement is done on the algorithm execution only. The parallel simulation is measured from before the start of the first thread until all threads have finished. During the measurements a fixed frame step size is used to step the simulation, independent of application performance. Using the measurements for all runs, the per-character and per-leggedness performance mean value (milliseconds) is calculated. The set-up template for how the parameters affect the end measurement values of the serial scenario is showcased in Figure 4.1.

The parallel scenario consists of the same variable parameters as the serial scenario but it will include one additional parameter: the *parallel loop invocation count*. The parallel loop invocation count is based upon the amount of available logical cores on the hardware on which the experiment is executed. The parallel loop invocation count ranges from 2 to 4. The set-up of the parallel scenario is showcased in Figure 4.3.

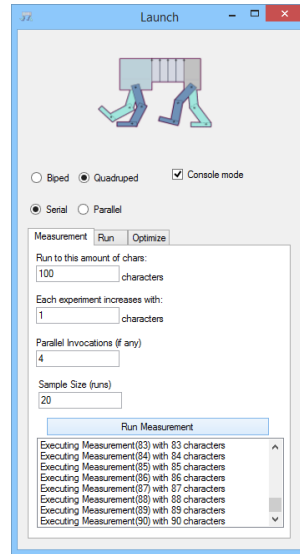


Figure 4.4: *The experiment automation application. This application executes the main program with the correct parameters. It will launch the main application several times, with incrementing parameters. It only runs one instance of the main application at a time.*

4.2 Technology

The experiment was executed on a computer with the following specifications:

- CPU: Intel Core i7 4700HQ: 2.4GHz, 8 logical cores, 4 physical cores
- RAM: 12GB DDR3
- GPU: Nvidia GeForce GTX 760M
- Operating System: Windows 8.1
- Language: C++
- Compiled using Visual Studio 2013

The main experiment application was written in C++. The measurements were performed using the high resolution *QueryPerformanceCounter*¹ from the Windows API. The main application can be launched with the previously mentioned variable parameters set by an external text file. The main application performs the set number of simulation runs before recording the mean and standard deviation. For each new initiated run the BulletPhysics engine is restarted to get deterministic outcomes in the states of the physics engine. To ensure that

¹[http://msdn.microsoft.com/en-us/library/windows/desktop/dn553408\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/dn553408(v=vs.85).aspx), Accessed 2014-12-20

the physics engine achieved deterministic states, the rigid body transformations were logged to text files, which were compared using the external diff-tool *TortoiseGitMerge*. In order to automate the tests an external tool was developed that schedules all executions of the main application consecutively with the correct variable parameters (see Figure 4.4). All measurements were written to disk by the main application (after the execution of all runs) that were then rendered to graphs using *GnuPlot*.

The memory consumption of the application when simulating one biped or one quadruped is about 10 MB when only executing the locomotion systems and about 40 MB for the whole program. When increasing the character count of the simulation to 100 the memory consumption is as follows: bipeds are 13 MB (whole application: 46 MB) and quadrupeds are 17 MB (whole application: 56 MB).

4.3 Validity Threats

The following validity threats have been considered for the experiment:

Other programs running in Windows might impact the performance measurement of the application. For the parallel simulation this might very well result in some threads taking longer to allocate if there is already too much parallel work from other applications. External applications have been disabled during the execution of this experiment. Default Windows system processes were not disabled. The experiment was executed as an 32-bit release build outside its development environment (Visual Studio).

The experiment when run in parallel might result in non-optimal workload distribution over threads for workloads that are not evenly divisible by the amount of threads. For these occurrences, the workload is increased by one for all threads $0 \leq i < n$ where $i < rest$. This proved to yield better performance for uneven workloads compared to letting a single thread allocate the whole rest-workload.

The current experiment scenarios only includes movement over flat terrain. As such, the results may not be representative for all real-time scenarios. The original algorithm has been proven to allow for locomotion over slightly uneven terrain [9], without further modifications. The phases in the algorithm is somewhat dependent on underlying terrain topology, as early foot strikes causes the state of the gait to change. However, these state changes still occur as a part of the natural cyclical nature of the gait.

There is a risk of wrongful interpretation of the end result measurement data. It might be interpreted as the combined performance of both the physics engine and the locomotion algorithm. As the physics engine is independent from the locomotion algorithm itself, it has not been measured. Only the locomotion algorithm has been measured.

The final results presented in Chapter 5 generated some peaks in standard

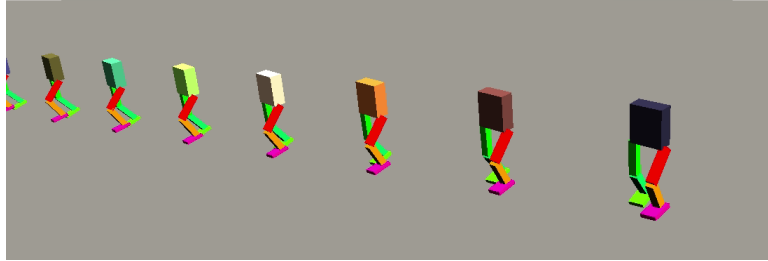


Figure 4.5: *Array of simultaneously simulated bipeds.*

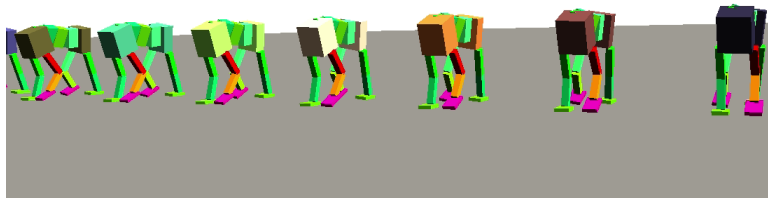


Figure 4.6: *Array of simultaneously simulated quadrupeds.*

deviation. These peaks are discussed in Chapter 6 showing them to be a random occurrence.

4.4 Summary

This chapter presented the method for evaluating the locomotion algorithm, an automated experiment that measures the execution time. The layout of the experiment was described. It presented the two experiment scenarios for bipeds and quadrupeds: the serial and the parallel scenario.

The variable parameters for the experiment are the character count (1 to 100) and the leg count (2 and 4). In the parallel scenario there is one additional parameter incrementing the parallel invocation count.

In Figure 4.5 and Figure 4.6 visualizations of multi-character simulations are shown, representing how the experiment will be run (except rendering) for several characters.

The chapter concluded with a description of the hardware used and a section on the validity threats. The results of this experiment are presented in Chapter 5.

Following the experiment a set of measurements were acquired. These measurements are presented in this chapter. An analysis of the results presented here are then presented in Chapter 6.

5.1 Observations

The results of the serial- and parallel scenarios for bipeds and quadrupeds were recorded from the performance measurement of each simulated frame. Line graphs showing the performance hit (algorithm execution time) in milliseconds as a function of character count were rendered and are presented in this section. The graphs were grouped based on character leg count.

5.1.1 Measurements

For each program execution, the number of characters were defined and the simulation of those characters were then executed several times to achieve a mean and its standard deviation, per simulated frame. An example of the performance over a whole simulation, frame-by-frame, for 20 biped characters can be seen in Figure 5.1. A frame-by-frame performance graph for 20 quadrupeds can be seen in Figure 5.2.

The mean and standard deviation of the whole simulation of a program execution in all program executions were stored and are presented in graphs in which algorithm execution time is presented as a function of character count. These graphs are also grouped on leggedness. In Figure 5.3 the serial and parallel performance of the biped character simulations are presented, for 1 to 100 simulated characters. A graph of the serial and parallel performance of 1 to 100 simulated quadruped characters are presented in Figure 5.4.

A graph representing the combined measurements for bipeds and quadrupeds can be seen in Figure 5.5. Note that the standard deviation have been omitted in this graph for clarity. Refer to Figure 5.3 (bipeds) and Figure 5.4 (quadrupeds) for an overview of the standard deviations.

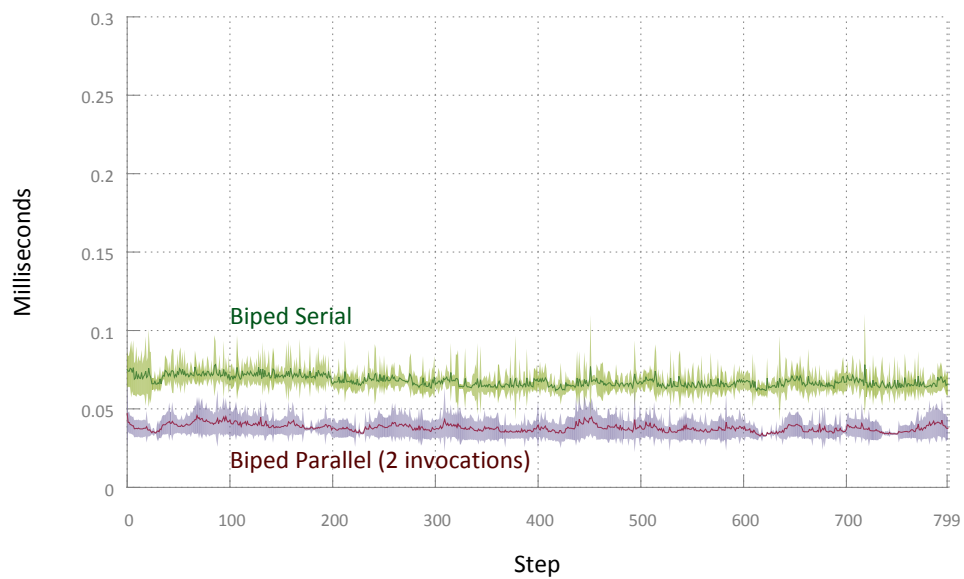


Figure 5.1: *Biped per-frame result example.* The performance measurements over 800 simulated frames for a single execution of 20 biped characters. The parallel measurement showcased is for two parallel invocations. When running the application synced at real-time speeds the shown simulation period corresponds to ~ 6.7 seconds. Standard deviation is shown as a filled graph for each case.

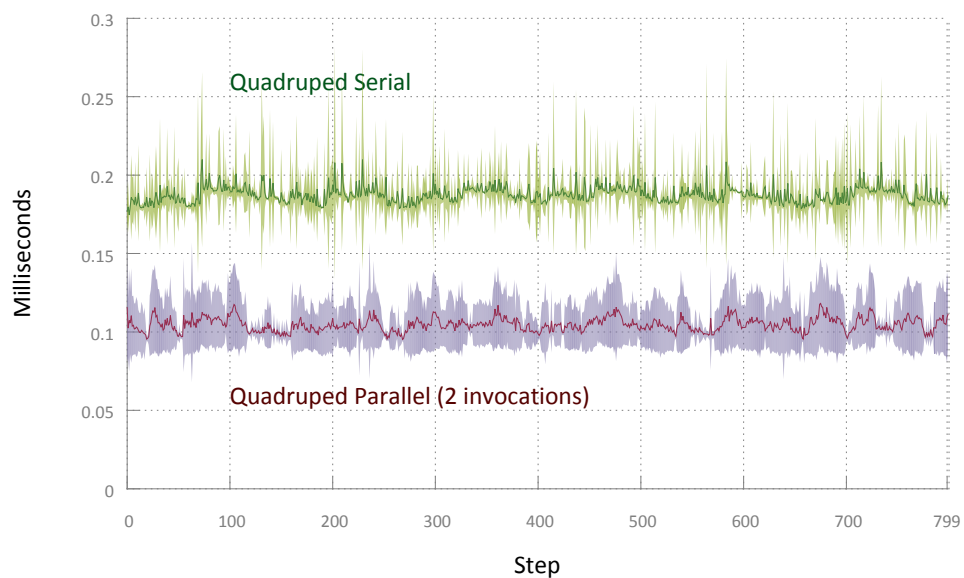


Figure 5.2: *Quadruped per-frame result example.* The performance measurements over 800 frames for a single execution of 20 quadruped characters. The showcased parallel measurement is for two parallel invocations. Standard deviation is shown as a filled graph for each case.

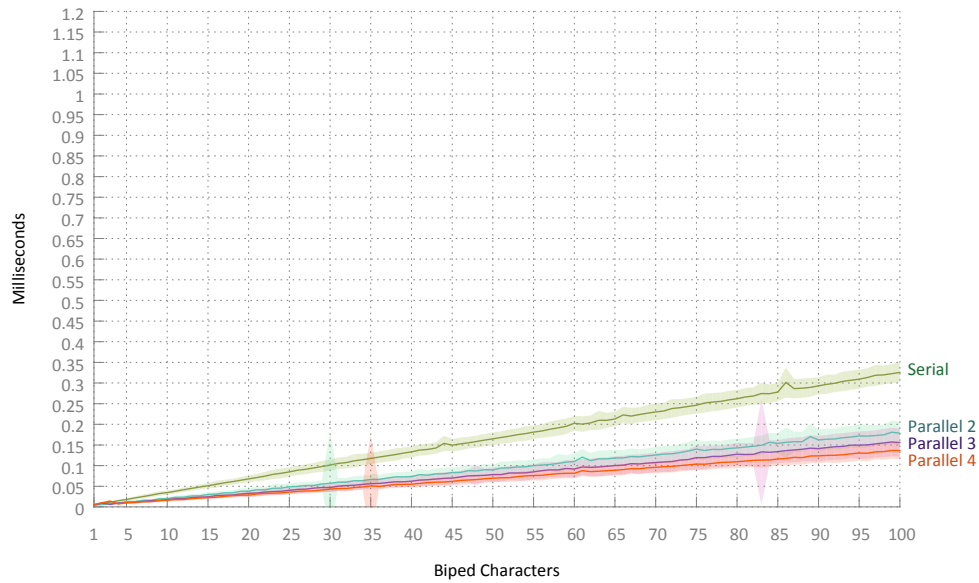


Figure 5.3: Biped per-character results. The performance results, in milliseconds (lower is better), of the biped characters. The measurements taken ranges from 1 to 100 characters. The graphs represent the serial (topmost) as well as the parallel (lower cluster) simulations of the bipeds. Standard deviation is shown as a filled graph for each case.

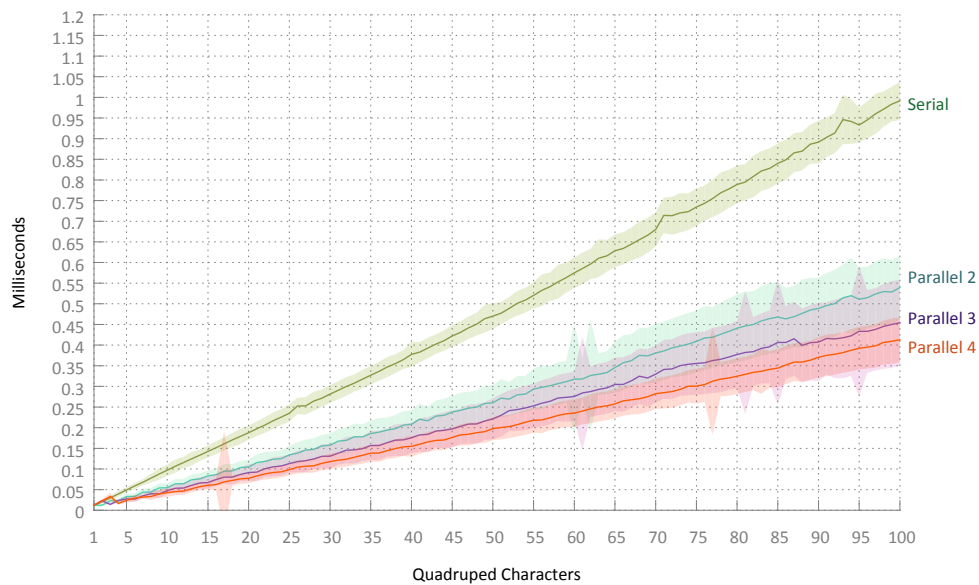


Figure 5.4: Quadruped per-character results. The performance results, in milliseconds (lower is better), of the quadruped characters. The measurements taken ranges from 1 to 100 characters. The graphs represent the serial- (topmost) as well as the parallel (lower cluster) simulations of the quadrupeds. Standard deviation is shown as a filled graph for each case.

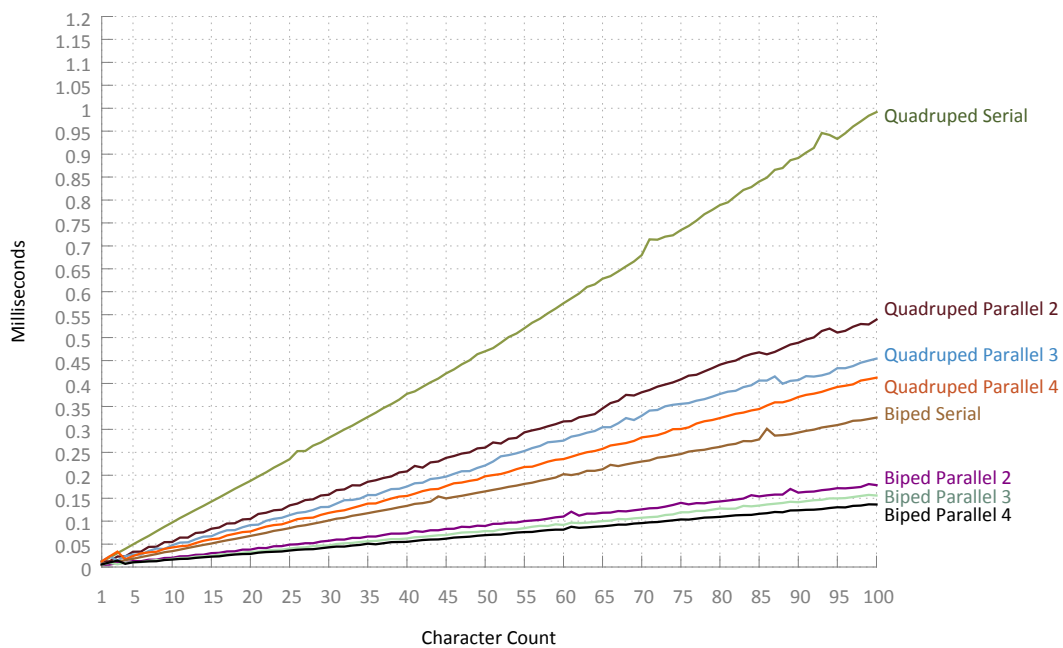


Figure 5.5: All per-character results. The average performance results (without standard deviation), in milliseconds (lower is better), of all characters. This graph represents the experiment as a whole, with results for both bipeds and quadrapeds, serial and parallel. This figure combines Figure 5.3 and Figure 5.4.

Provided with the measurement results, this chapter presents an analysis of this data. Also presented are the measurements in relation to each other.

6.1 Serial Results Analysis

The results for the biped serial simulation measurements ranges from a mean of ~ 0.004 ms for 1 character, to a mean of ~ 0.326 ms for 100 characters. The biped measurements have a ratio of about 1 : 75 for the mean when normalizing the range by division by the 1 character measurement.

For the quadruped, the serial measurements ranges from a mean of ~ 0.011 ms for 1 character, to a ~ 0.992 ms mean for 100 characters. The 1-character normalized ratios for the quadruped are about 1 : 91 for the mean.

An overview of these ratios are presented in Table 6.1. A comparison chart of the mentioned ratios for the mean values are visible in Figure 6.1, it gives an overview of the behaviour of the slowdown rate. The standard deviation increased with the character count in all measurements. It should be noted that the standard deviation exhibited peaks in various configurations, that were of larger magnitude than the 100 character setups. The quadruped had a $\sim 2.5 \times$ slower execution time than the biped when simulating a single character. When simulating 100 characters, the execution time of the quadrupeds were $\sim 3.0 \times$ slower than that of the bipeds.

Leggedness	1 character	100 characters	Execution time ratio
Biped	0.004 ms	0.326 ms	1 : 75
Quadruped	0.011 ms	0.992 ms	1 : 91

Table 6.1: *The measurements and ratios for the mean serial execution time for the biped. The ratios have been normalized to the 1-character simulation.*

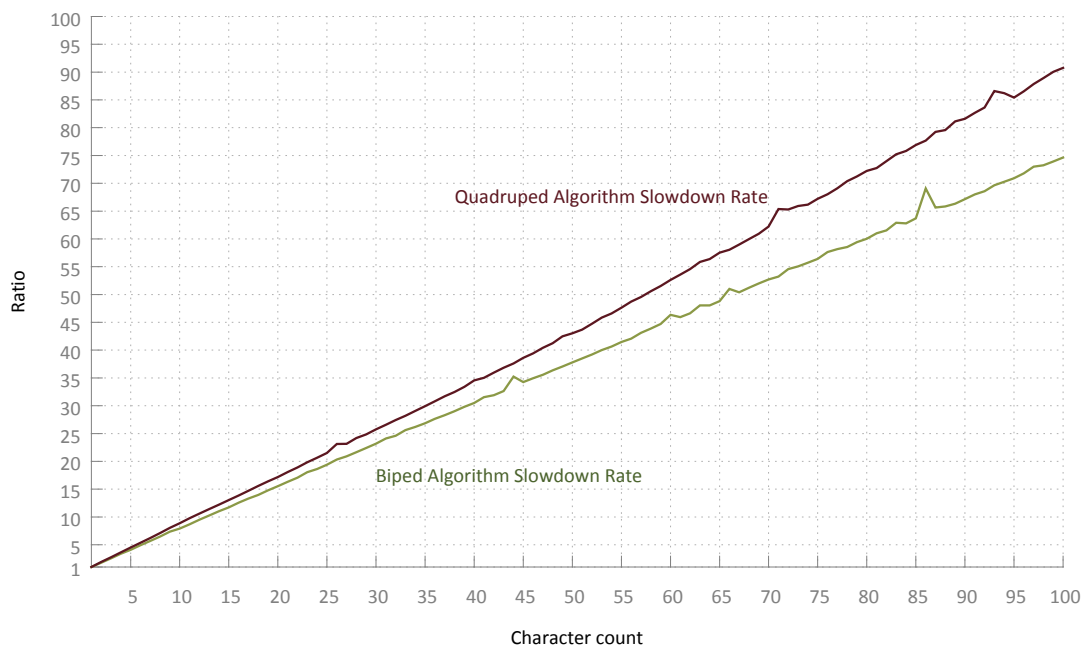


Figure 6.1: *Serial slowdown rate comparison (lower is better). Comparison view of the normalized performance ratio between serially simulated bipeds and quadrupeds. The graphs have been normalized to the respective 1-character simulation, for both cases.*

Threads	1 character	100 characters	Execution time ratio
2	0.005 ms	0.178 ms	1 : 36
3	0.005 ms	0.156 ms	1 : 32
4	0.006 ms	0.136 ms	1 : 24

Table 6.2: *The measurements and ratios for the mean parallel execution time for the biped. The ratios have been normalized to the 1-character simulation.*

Threads	1 character	100 characters	Execution time ratio
2	0.012 ms	0.539 ms	1 : 46
3	0.012 ms	0.454 ms	1 : 37
4	0.012 ms	0.413 ms	1 : 33

Table 6.3: *The measurements and ratios for the mean parallel execution time for the quadruped. The ratios have been normalized to the 1-character simulation.*

6.2 Parallel Results Analysis

The resulting ratios for the biped parallel simulation measurements, from 1 to 100 characters, are presented in Table 6.2. The resulting ratios for the quadruped parallel simulation measurements, are presented in Table 6.3.

The values in these tables are not a perfect representation of the slowdown behaviour as workload distribution matters in the parallel simulation. For a more complete overview, all the parallel ratios are also presented in Figure 6.2.

For the 100-character parallel simulation, the quadrupeds were $\sim 3.0 \times$ slower for 2 threads, $\sim 2.9 \times$ slower for 3 threads and $\sim 3.0 \times$ slower for 4 threads.

In the parallel measurements the standard deviation also increased alongside the character count. As well as in the serial results, there were peaks for some configurations.

6.3 Trends and Relations

The aforementioned ratios gives an overview of the relative efficiency of increasing the workload. A ratio of 1:100 is a perfect linear slowdown, showcasing no increase in relative efficiency. All the current measurements got a lower ratio, the quadruped serial execution being the closest to a perfectly linear slowdown. The parallel implementations achieved lower ratios, with the most drastic change between serial and two parallel invocations, with diminishing results relative to the initial when increasing invocation count further.

The difference between serial- and parallel measurements were statistically significant for both bipeds and quadrupeds for more than 5 characters, as the standard deviation areas as seen in Figure 5.3 and Figure 5.4 (Chapter 5) does not overlap for this range. There was varying overlap of the standard deviation areas

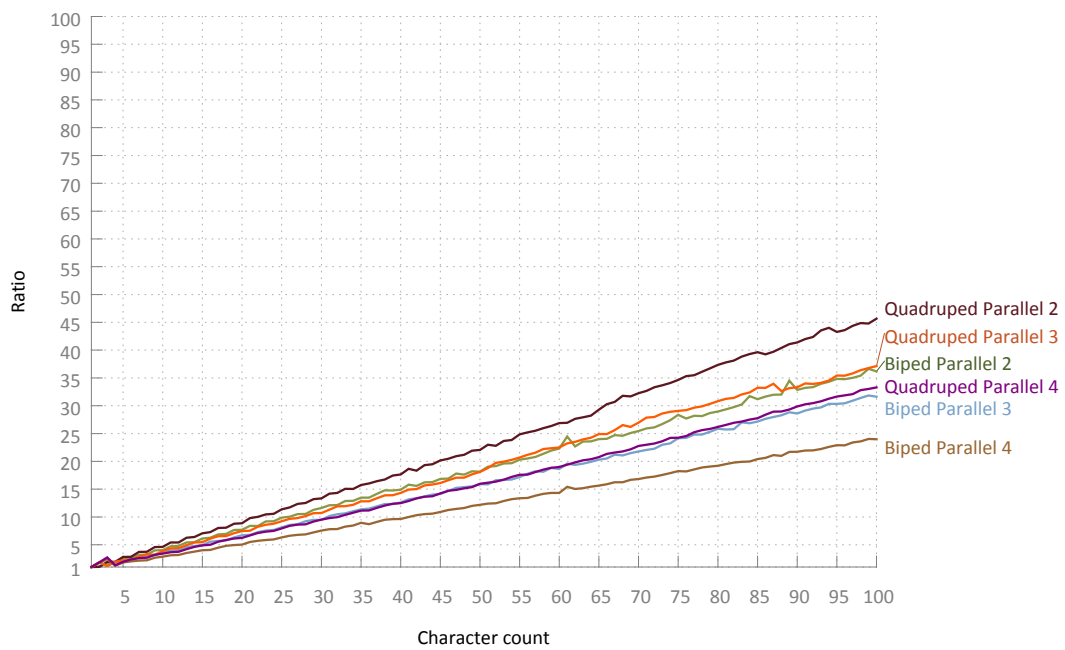
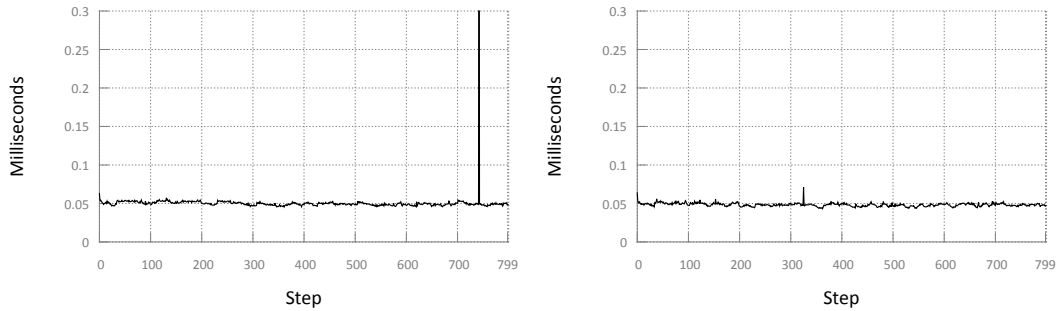


Figure 6.2: Parallel slowdown rate comparison (lower is better). Comparison view of the normalized performance ratio between parallel simulations of bipeds and quadrupeds. The graphs have been normalized to the respective 1-character simulation, for all cases.



(a) Simulation of 35 biped characters for 4 parallel invocations. Execution time peak visible between frame 700 and 799.

(b) Second simulation of 35 biped characters for 4 parallel invocations. No peak between frame 700 and 799 occurred.

Figure 6.3: Random outliers in the per-frame measurement. Both graphs represent the simulation of 35 biped characters over 4 threads. In graph (a) a large peak is visible near the end of the simulation. The second graph (b) represents a re-run of the same settings.

within the group of parallel measurements for bipeds and quadrupeds respectively.

Performance difference based on leg count is statistically significant for the serial result over the whole range, without any standard deviation overlap. This is true when comparing the parallel measurements between leg counts as well, but only for more than 5 characters and with the exception of three peaks registered within the inclusive range of 17 to 35 characters. These peaks were caused by random outliers in the per-frame measurement, where one frame exhibited a peak in the execution time. Re-runs of the experiment shows that the peaks appear at random, as seen in Figure 6.3. All average measurements and their standard deviations have been combined in Figure 6.4.

The trend of the results show an almost linear increase, with only a very slight overall upwards curvature, most discernible in the serial simulation for the quadruped. A comparison of the quadruped simulation results and a linear trend graph, fitted using linear regression is shown in Figure 6.5.

6.4 Summary

This chapter provided an analysis of the measurement data provided from the results. It presented additional graphs of the slowdown rate of all measurements. In Chapter 7 the thesis is concluded with discussion of the results and possibilities for future work. It summarises the work and contribution as well as provides comparisons to related work.

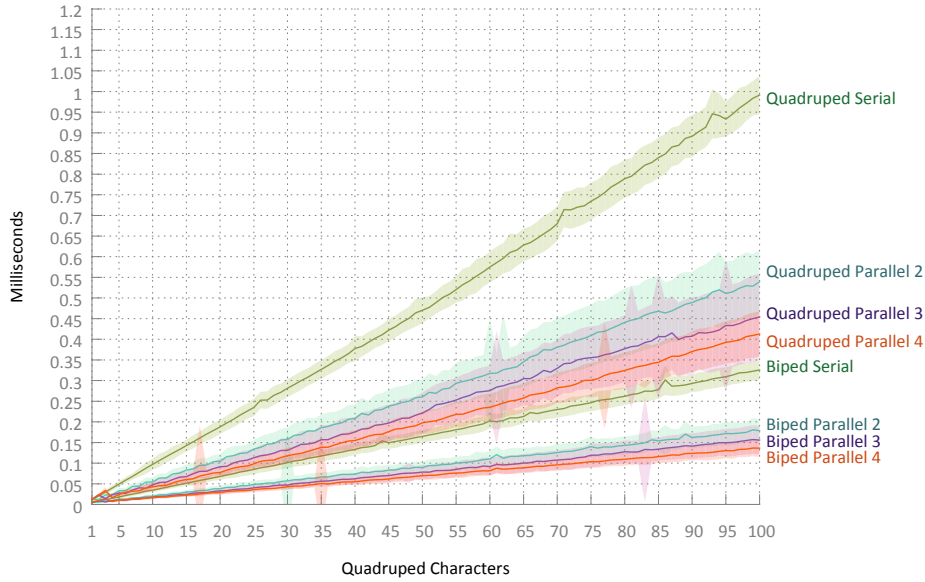
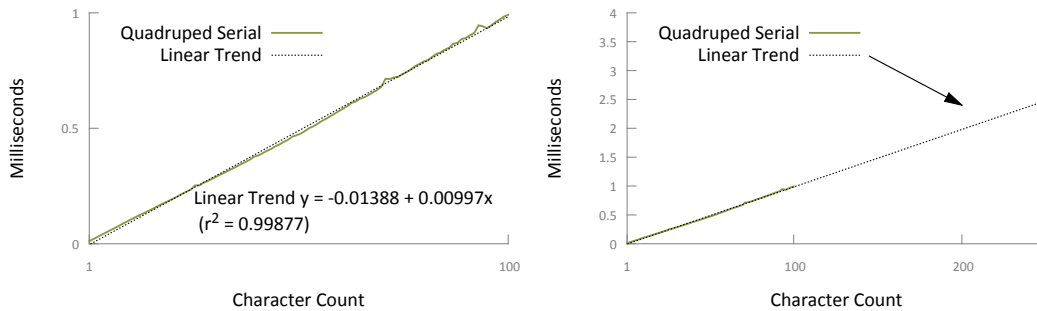


Figure 6.4: Combination of Figure 5.3 and Figure 5.4.



(a) Trend estimation on Quadruped serial simulations.

(b) Overview of a larger range, 1-250 characters. Trend estimation on Quadruped serial simulations.

Figure 6.5: Linear trend for the results on the serial Quadruped simulations. The estimation and the coefficient of determination was calculated using linear regression on the execution time data by the use of the GnuPlot stats-function. The left plot (a) presents the measurement and the linear trend, while the right plot (b) presents the same graphs with a view of a larger range.

Chapter 7

Conclusions and Future Work

In this thesis the concept of procedural animation of locomotion was presented and an experiment was set up to measure the performance relations for such an animation model, single- and multi-threaded. This chapter presents a summary followed by conclusions that can be drawn from the results presented in Chapter 5 and the analysis in Chapter 6. Concluding this chapter and this thesis are summaries of the contribution this work provides as well as the experience gained from it, and how it may be utilized and advanced in the future.

7.1 Thesis Summary

The thesis presents an implementation of an algorithm for procedural animation of locomotion and outlines an experiment to measure the performance relations between animation of bipedal- and quadrupedal characters, using the procedural algorithm.

The thesis begins by presenting the concept of procedural animation, and more specifically, how it can be used to produce animation of terrestrial locomotion. Various methods to accomplish this were presented, both kinematic- and dynamics-driven variants. A more in-depth background was then presented of the dynamics-driven (physics driven) method to be implemented for performance measurement.

The specifics of the algorithm implementation were then presented in Chapter 3. The algorithm allows for animating both bipeds and quadrupeds. Through the use of virtual forces the algorithm approximates a nullification of gravity by calculating countering per-joint torques. Virtual forces also produces per-joint torques needed in the model to allow for forwards motion of the character. PD controllers are further used to control internal shape of the legs (knee placement) and spine strength (in the quadruped character). The algorithm was implemented for both serial- and parallel (on CPU) execution.

Using a greedy off-line optimization phase, the input parameters to the algorithm are tweaked based on the desired gait and the character anatomy. The finished optimized parameters then allow for real-time execution of the animation algorithm.

The experiment set up to evaluate performance relations focused only on the execution time for the algorithm itself. It executed 800 steps of simulation (about 7 footsteps) for 1 to 100 characters. Both bipeds and quadrupeds were evaluated on both serial- and parallel simulations. The experiment was automated using a separate application which was custom made for the purpose.

7.2 Method and Implementation Discussion

The animation algorithm itself proved to have a relatively low performance impact compared to the application as a whole. The relationship in performance between bipeds, quadrupeds and between serial- and parallel execution are perhaps not that surprising. What might be surprising however is the somewhat low performance impact, peaking at about 1 ms for 100 quadrupeds. Given this result, it might be more interesting to evaluate the overall impact of the locomotion algorithm as well as the resulting solving done by the physics engine.

Automating the experiments proved useful due to the sheer amount of resulting simulation time needed when running several simulations to get mean- and standard deviation values.

The locomotion algorithm chosen proved difficult to get working adequately. Generating plausible and working gaits using it is somewhat cumbersome. The character anatomy and the constant PD gains have to be carefully planned out. Even though many parameters are automatically optimized, the current local optimization does require starting parameters which are somewhat plausible. Other optimization algorithms were not tested in this thesis, but other algorithms, such as covariance matrix adaptation have been applied successfully in previous works as well to optimize parameters for locomotion [14].

The gait can be optimized without motion capture data, in this thesis the result of the IK solvers were used as a simplified substitute (to reward bending of the legs), for instance. However, the resulting quality has not been analysed.

One should expect the optimization and design of gaits to be quite time consuming. Finding robust gaits is especially time consuming. Given correct starting conditions, optimization took about 15-30 minutes (100-300 generations) for the characters in this thesis. For better results, about 3-4 optimizations were ran with incrementing lengths, on its preceding results. However, the characters in this thesis were not optimized for turbulence and thus easily trips and falls over when interacted with. More thorough optimizations have been recorded to take about 2-12 hours [9, 13, 14].

When choosing similar methods for implementation in for example games, one should carefully consider if the extra amount of time needed is worth it for the extra realism that a physics-driven propulsive locomotion method gives. Especially the extra amount of work needed by those tweaking the gait and making it robust. A simple character redesign could result in hours of optimizing work, which in

the worst case might not even find a pleasing gait. A kinematically-driven method might be sufficient in many cases.

The method tests 1-100 characters. The results show an almost linear slowdown in this range with just a slight inclination upwards.

The number of animated characters in an actual game will vary depending on the game. The game *Madden NFL 10* by *Electronic Arts* for instance, reported the need to animate about 100 characters on the field in real-time [18]. The technical solution used to animate a character in a game might vary as well depending on its current *level-of-detail* (LOD) setting [33]. LOD in turn may for instance depend on rendering distance or run-time performance. Which means that for a physically simulated locomotion solution, a situation were 100 characters or more are displayed, a LOD system could determine that only some need to be physically simulated, while others use less intensive methods. The results presented in this thesis shows that it is possible, performance-wise, to simulate locomotion for a similar amount of characters with a pose-driven feedback method, in real-time.

7.3 Conclusions

The research question posed in Chapter 1 queried what real-time performance relations could be observed between a single-threaded and multi-threaded execution of a pose-driven feedback locomotion algorithm, for groups of characters and for varying amounts of legs.

Given the experiment results, we can discern that slowdown will scale almost linearly for 1-100 characters. The quadrupedal slowdown rate (when increasing character count) is somewhat higher than the slowdown rate of the biped. The simulation of the quadruped is also overall more performance intensive, taking roughly 3 times longer to compute. This might be due to the extra force calculations needed for the spine (4 spine links for this experiment).

Multi-threading of the algorithm as done in this thesis provided almost a halving of the execution time when going from serial execution to 2 parallel invocations, in the case of 100 characters (both quadrupeds and bipeds), with diminishing returns for further increasing of the parallel invocations. Although the parallelization do allow for performance improvements of the algorithm; given the context presented in this thesis, a parallelization on the scope done here is questionable in its efficiency. Larger bottlenecks are already apparent, and workload distribution could probably be done in another manner for better overall efficiency.

7.4 Contribution and Comparisons

This thesis presented a first thorough look into the performance aspects of a typical pose-driven feedback procedural animation algorithm for physics-driven locomotion. Compared to previous works, relation in execution time between leggedness and character counts are presented, as well as an analysis of the execution time slowdown rates.

These measurements provide a basis of comparison for further performance evaluations of similar algorithms. The results presented can also be used to decide the relevance and usefulness of implementing this algorithm.

A novel implementation was made which generalized existing biped- and quadruped locomotion simulation solutions to allow for both, using the same simulator. The source code for the implementation has been made available as open source¹ for further studies or derivative works.

7.5 Future Work

As the performance impact of the algorithm for the evaluated population count of characters is relatively low, it would be of interest to evaluate the overall impact of the locomotion algorithm combined with the physics solver. For such an experiment it would be of interest to compare the impact and differences for several available physics solvers.

Given a context other than real-time interactive applications, or given environments with smaller physics bottlenecks, it might also be motivated to measure much larger simulated character population counts as well.

This thesis evaluated simple bipeds and quadrupeds, but it might be interesting to broaden this scope to include insects, spiders or fantasy characters with any leg count. Alien anatomies might even call for branching limbs, something that no doubt could open up new interesting challenges to find physically simulated gaits for.

It could also be interesting to evaluate what results a combined locomotion- and physics solver might give. To find out what optimizations could be made in a specialized context like that and if it provided differing results. Such a solution might be adaptable for GPGPU execution and evaluation as well, as a combined locomotion- and physics compute-solution might lessen the amount of read- and write operations to the GPU which might have been a bottleneck for a GPGPU dynamics-driven algorithm with a standalone physics solver.

This thesis covered only one type of physics-driven locomotion algorithm. There exist others for which performance data is also lacking, such as the alternatives presented in Chapter 2. One could perform a more extensive quality- and

¹www.github.com/jarllarsson/promenade (Made available 2015-02-08)

performance evaluation, over several different algorithms, along with a measurement in implementation and optimization complexity.

Finally, what would be really interesting, is to actually successfully implement a dynamics-driven propulsive locomotion system in a game or other interactive application. A game with a similar premise like *Spore* for example, in which the user can design weird and wonderful fantasy creatures, and then see them come to life. To simulate such creatures using physics, without too much failure and within the strict time scopes of a game is a complex but alluring task.

References

- [1] R. M. N. Alexander, *Principles of Animal Locomotion*. Princeton University Press, 2003.
- [2] G. M. Amdahl, “Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities,” in *Proc AFIPS Conf*, vol. 30. AFIPS Press, 1967, pp. 483–485.
- [3] D. H. Bailey, “Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers,” 1991.
- [4] F. G. Barth, *A spider’s world: senses and behavior*. Springer Science and Business Media, 2002.
- [5] J. Bates, “The role of emotion in believable agents,” *Communications of the ACM*, vol. 37, no. 7, pp. 122–125, 1994.
- [6] C. M. Biancardi and A. E. Minetti, “Biomechanical determinants of transverse and rotary gallop in cursorial mammals.” *The Journal of experimental biology*, vol. 215, no. Pt 23, pp. 4144–4156, Dec. 2012.
- [7] A. Boeing and T. Bräunl, “Evaluation of Real-time Physics Simulation Systems,” in *Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia*, vol. 1, no. 212. Perth, Australia: ACM, 2007, pp. 281–288.
- [8] S. Coros, P. Beaudoin, and M. van de Panne, “Generalized Biped Walking Control,” *ACM Transactions on Graphics*, vol. 29, no. 4, p. 1, Jul. 2010.
- [9] S. Coros, A. Karpathy, B. Jones, L. Reveret, and M. van de Panne, “Locomotion Skills for Simulated Quadrupeds,” *ACM Transactions on Graphics (TOG)*, vol. 30, no. 4, p. 59, 2011.
- [10] A. Darte and G.-R. Perrin, *The Data Parallel Programming Model*. Springer Berlin / Heidelberg, 1996.
- [11] J. Fang, C. Joang, and D. Terzopoulos, “Modeling and Animating Myriapoda : A Real-Time Kinematic / Dynamic Approach,” no. i, pp. 203–212, 2007.

- [12] B. R. Gaster, L. Howes, and A. M. Devices, “Can GPGPU Programming Be Liberated from the Data-Parallel Bottleneck?” *Computer*, vol. 45, no. 8, pp. 42–52, 2012.
- [13] T. Geijtenbeek, *Animating Virtual Characters using Physics-Based Simulation*, 2013.
- [14] T. Geijtenbeek, M. van de Panne, and a. F. van der Stappen, “Flexible muscle-based locomotion for bipedal creatures,” *ACM Transactions on Graphics*, vol. 32, no. 6, pp. 1–11, 2013.
- [15] T. M. Griffin, R. P. Main, and C. T. Farley, “Biomechanics of quadrupedal walking: how do four-legged animals achieve inverted pendulum-like movements?” *The Journal of experimental biology*, vol. 207, no. Pt 20, pp. 3545–58, Sep. 2004.
- [16] J. L. Gustafson, “Reevaluating Amdahl’s law,” *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, May 1988.
- [17] C. Gyrling, “Creating a Character in Drake’s Fortune,” www.naughtydog.com/docs/Naughty-Dog-GDC08-Creating-a-Character-in-Drakes-Fortune.pdf (Accessed 2014-03-16), 2008.
- [18] J. Harmon, “Football at 60 fps: The Challenges of Rendering Madden NFL 10,” <http://www.gdcvault.com/play/1012475/Football-at-60-fps-The> (Accessed 2015-02-05).
- [19] C. Hecker, B. Raabe, R. W. Enslow, J. DeWeese, J. Maynard, and K. van Prooijen, “Real-time Motion Retargeting to Highly Varied User-Created Morphologies,” in *Proceedings of ACM SIGGRAPH ’08*, 2008.
- [20] E. S. L. Ho, H. P. H. Shum, Y.-M. Cheung, and P. C. Yuen, “Topology Aware Data-Driven Inverse Kinematics,” *Computer Graphics Forum*, vol. 32, no. 7, 2013.
- [21] R. S. Johansen, “Automated Semi-Procedural Animation for Character Locomotion,” Master Thesis, Aarhus Universitet, Institut for Informations- og Medievidenskab, 2009.
- [22] A. A. Karim, T. Gaudin, A. Meyer, A. Buendia, and S. Bouakaz, “Procedural locomotion of multilegged characters in dynamic environments,” *Computer Animation and Virtual Worlds*, vol. 24, no. 1, pp. 3–15, 2013.
- [23] B. Kenwright, “Generating Responsive Life-Like Biped Characters,” 2012.

- [24] B. Kenwright, “Responsive Biped Character Stepping: When Push Comes to Shove,” *2012 International Conference on Cyberworlds*, pp. 151–156, Sep. 2012.
- [25] B. Kenwright, “Controlled 3D Biped Stepping Animations Using the Inverted Pendulum and Impulse Constraints,” *2013 International Conference on Cyberworlds*, pp. 326–329, Oct. 2013.
- [26] B. Kenwright, “Real-Time Dynamic Multi-Character Agents,” 2013.
- [27] B. Kenwright, “Real-Time Reactive Biped Characters Staying Upright and Balanced,” *Transactions on Computational Science XVIII*, pp. 155–171, 2013.
- [28] B. Kenwright, R. Davison, and G. Morgan, “Dynamic Balancing and Walking for Real-time 3D Characters,” 2011.
- [29] J.-Y. Kim, I.-W. Park, and J.-H. Oh, “Walking Control Algorithm of Biped Humanoid Robot on Uneven and Inclined Floor,” *Journal of Intelligent and Robotic Systems*, vol. 48, no. 4, pp. 457–484, Jan. 2007.
- [30] J.-P. Kivistö, “Real-Time Muscle Animation,” Thesis, Kajaanin ammattikorkeakoulu, 2012.
- [31] V. Kodaganallur and A. K. Sen, “Greedy by Chance - Stochastic Greedy Algorithms,” *2010 Sixth International Conference on Autonomic and Autonomous Systems*, pp. 182–187, Mar. 2010.
- [32] J. Lee, M. Samadi, Y. Park, and S. Mahlke, “Transparent CPU-GPU collaboration for data-parallel kernels on heterogeneous systems,” *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, pp. 245–255, Sep. 2013.
- [33] G. Lindahl and M. A. Johansson, “LOD Techniques for Boosting Rendering Performance,” <http://www.gdcvault.com/play/1017872/LOD-Techniques-for-Boosting-Rendering> (Accessed 2015-02-05), 2013.
- [34] W. D. Lister and A. Day, “Stream-based animation of real-time crowd scenes,” *Computers and Graphics*, vol. 36, no. 6, pp. 651–657, Oct. 2012.
- [35] C. Liu, “An Analysis of the Current and Future State of 3D Facial Animation Techniques and Systems,” Master thesis, Simon Fraser University, 2009.
- [36] L. Liu, “Terrain Runner : Control , Parameterization , Composition , and Planning for Highly Dynamic Motions,” *ACM Transactions on Graphics (TOG)*, vol. 31, 2012.

- [37] T. Macintosh, “Animation and Player Control in Uncharted 1 and 2,” www.gdcvault.com/play/1012300/Animation-and-Player-Control-in (Accessed 2014-02-13), 2010.
- [38] I. S. MacKenzie and C. Ware, “Lag As a Determinant of Human Performance in Interactive Systems,” in *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, ser. CHI '93. New York, NY, USA: ACM, 1993, pp. 488–493.
- [39] T. Mclaughlin, L. Cutler, and D. Coleman, “Character Rigging , Deformations , and Simulations in Film and Game Production,” in *ACM SIGGRAPH 2011 Courses*, Vancouver, British Columbia, Canada, 2011.
- [40] R. P. Paul, *Robot Manipulators: Mathematics, Programming, and Control : the Computer Control of Robot Manipulators*, ser. Artificial Intelligence Series. MIT Press, 1981.
- [41] K. Perlin, “Creating emotive responsive characters within virtual worlds,” *Lecture Notes in Artificial Intelligence*, vol. 1834, pp. 99–106, 2000.
- [42] K. Perlin and A. Goldberg, “Improv : A System for Scripting Interactive Actors in Virtual Worlds,” *Acm Siggraph*, pp. 205–216, 1996.
- [43] J. Pratt, “Virtual Model Control: An Intuitive Approach for Bipedal Locomotion,” *The International Journal of Robotics Research*, vol. 20, no. 2, pp. 129–143, Feb. 2001.
- [44] N. Pronost and A. F. V. D. Stappen, “Simple Data-Driven Control for Simulated Bipeds,” *SCA '12: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 211–219, 2012.
- [45] M. H. Raibert and J. K. Hodgins, “Animation of dynamic legged locomotion,” *ACM SIGGRAPH Computer Graphics*, vol. 25, no. 4, pp. 349–358, 1991.
- [46] C. F. Rose III, P.-P. J. Sloan, and M. F. Cohen, “Artist-Directed Inverse-Kinematics Using Radial Basis Function Interpolation,” *Computer Graphics Forum*, vol. 20, no. 3, pp. 239–250, Sep. 2001.
- [47] D. Rosen, “Animation Bootcamp: An Indie Approach to Procedural Animation,” <http://www.gdcvault.com/play/1020049/Animation-Bootcamp-An-Indie-Approach> (Accessed 2015-01-31), 2014.
- [48] A. Shapiro, “Building a character animation system,” *Motion in Games*, pp. 98–109, 2011.

- [49] A. Shoulson and N. Marshak, “ADAPT : The Agent Development and Prototyping Testbed,” in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, vol. 1, no. 212. Orlando, Florida: ACM, 2013, pp. 9–18.
- [50] K. Sims, “Evolving Virtual Creatures,” in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, no. July. ACM, 1994, pp. 15–22.
- [51] S. Singh, M. Kapadia, G. Reinman, and P. Faloutsos, “Footstep navigation for dynamic crowds,” *Computer Animation and Virtual Worlds*, vol. 22, no. April, pp. 151–158, 2011.
- [52] C. Sunada, D. Argaez, S. Dubowsky, and C. Mavroidis, “A coordinated Jacobian transpose control for mobile multi-limbed robotic systems,” *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pp. 1910–1915, 1994.
- [53] Y.-Y. Tsai, W.-C. Lin, K. B. Cheng, J. Lee, and T.-Y. Lee, “Real-time physics-based 3D biped character animation using an inverted pendulum model.” *IEEE transactions on visualization and computer graphics*, vol. 16, no. 2, pp. 325–37, 2010.
- [54] M. Unuma, K. Anjyo, and R. Takeuchi, “Fourier Principles for Emotion-based Human Figure Animation Munetoshi Unuma.”
- [55] H. Van Welbergen, B. J. H. Van Basten, a. Egges, Z. M. Ruttkay, and M. H. Overmars, “Real Time Animation of Virtual Humans: A Trade-off Between Naturalness and Control,” *Computer Graphics Forum*, vol. 29, no. 8, pp. 2530–2554, Dec. 2010.
- [56] C. L. Vaughan, B. L. Davis, and J. C. O’Connor, *Dynamics of Human Gait*. Kiboho Publishers, 1999.
- [57] J. M. Wang, S. R. Hamner, S. L. Delp, and V. Koltun, “Optimizing Locomotion Controllers Using Biologically-based Actuators and Objectives,” *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, pp. 25:1—25:11, 2012.
- [58] C. Welman and B. S. Simon, “Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation,” no. September, 1993.
- [59] Z. Xiao and J. J. Zhang, “Motion Data Correction and Extrapolation Using Physical Constraints,” *Ninth International Conference on Information Visualisation (IV’05)*, pp. 571–576, 2005.

- [60] K. K. Yin, K. Loken, and M. van de Panne, "SIMBICON: Simple Biped Locomotion Control," 2007.