ON-LINE LOCOMOTION SYNTHESIS FOR VIRTUAL HUMANS

THÈSE N^{O} 3431 (2005)

PRÉSENTÉE À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

Institut des systèmes informatiques et multimédias

SECTION D'INFORMATIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Pascal GLARDON

ingénieur informaticien diplômé EPF
de nationalité suisse et originaire de Cugy (Fribourg)

acceptée sur proposition du jury:

Prof. D. Thalmann, directeur de thèse
Dr K. Aminian, rapporteur
Dr R. Boulic, rapporteur
Prof. T. Vetter, rapporteur
Dr X. Wang, rapporteur

Lausanne, EPFL
2006

To my parents
To Sonia

”Le bloc de granit
qui était un obstacle sur le chemin du faible
devient une marche sur le chemin du fort.”

Thomas Carlyle

Abstract

Ever since the development of Computer Graphics in the industrial and academic worlds in the seventies, public knowledge and expertise have grown in a tremendous way, notably because of the increasing fascination for Computer Animation. This specific field of Computer Graphics gathers numerous techniques, especially for the animation of characters or virtual humans in movies and video games. To create such high-fidelity animations, a particular interest has been dedicated to motion capture, a technology which allows to record the 3D movement of a live performer. The resulting realism motion is convincing. However, this technique offers little control to animators, as the recorded motion can only be played back. Recently, many advances based on motion capture have been published, concerning slight but precise modifications of an original motion or the parameterization of large motion databases. The challenge consists in combining motion realism with an intuitive on-line motion control, while preserving real-time performances.

In the first part of this thesis, we would like to add a brick in the wall of motion parameterization techniques based on motion capture, by introducing a generic motion modeling for locomotion and jump activities. For this purpose, we simplify the motion representation using a statistical method in order to facilitate the elaboration of an efficient parametric model. This model is structured in hierarchical levels, allowing an intuitive motion synthesis with high-level parameters. In addition, we present a space and time normalization process to adapt our model to characters of various sizes.

In the second part, we integrate this motion modeling in an animation engine, thus allowing for the generation of a continuous stream of motion for virtual humans. We provide two additional tools to improve the flexibility of our engine. Based on the concept of motion anticipation, we first introduce an on-line method for detecting and enforcing foot-ground constraints. Hence, a straight line walking motion can be smoothly modified to a curved one. Secondly, we propose an approach for the automatic and coherent synthesis of transitions from locomotion to jump (and inversely) motions, by taking into account their respective properties.

Finally, we consider the interaction of a virtual human with its environment. Given initial and final conditions set on the locomotion speed and foot positions, we propose a method which computes the corresponding trajectory. To illustrate this method, we propose a case study which mirrors as closely as possible the behavior of a human confronted with an obstacle: at any time, obstacles may be interactively created in front of a moving virtual human. Our method computes a trajectory allowing the virtual human to precisely jump over the obstacle in an on-line manner.

Résumé

Le développement de l'infographie dans les mondes industriel et académique a soutenu la croissance des connaissances et de l'expertise publique, notamment en raison d'une fascination grandissante pour l'animation par ordinateur. Ce domaine spécifique à l'infographie regroupe de nombreuses techniques, surtout dédiées à l'animation de personnages ou d'humains virtuels pour la réalisation de films de cinéma et de jeux vidéo. Pour la création d'animation de haute qualité, la capture de mouvement, une technique permettant d'enregistrer en direct les mouvements 3D d'un acteur, s'est révélée porteuse. Le mouvement ainsi reproduit bénéficie d'un réalisme convainquant. Cependant, peu de contrôle est offert aux animateurs professionnels, le mouvement ne pouvant qu'être joué en boucle. Récemment, de nombreuses solutions basées sur la capture du mouvement ont été publiées, que ce soit pour apporter des modifications fines mais précises au mouvement capturé, ou pour paramétrer une base de données d'enregistrements. La difficulté réside dans l'obtention d'un contrôle intuitif et en temps réel des paramètres du mouvement, tout en préservant le réalisme original dans l'animation produite.

Dans la première partie de cette thèse, nous tentons d'apporter une pierre à l'édifice des techniques de paramétrisation d'animation basées sur la capture du mouvement en introduisant un modèle de mouvement générique pour des activités de locomotion et de saut. Dans ce but, nous simplifions la représentation du mouvement en utilisant une méthode statistique, facilitant l'élaboration d'un modèle paramétrique efficace. Ce modèle, structuré en niveaux hiérarchiques, permet une synthèse du mouvement de manière intuitive à l'aide de paramètres de haut-niveau. De plus, nous présentons une normalisation en espace et en temps afin d'adapter notre modèle à des personnages de tailles diverses.

Dans la deuxième partie, nous intégrons ce modèle de mouvement dans un moteur d'animation, permettant ainsi la génération d'un flux continu de mouvements pour des humains virtuels. Nous proposons deux outils pour améliorer la flexibilité de notre moteur, tout en conservant des performances en temps réel. En utilisant le concept d'anticipation du mouvement, nous introduisons tout d'abord une méthode de détection et d'application de contraintes entre pied et sol. Ainsi, un mouvement de marche en ligne droite peut être modifié pour suivre une courbe, ceci sans discontinuité. Le deuxième outil permet de synthétiser des transitions de manière automatique et cohérente entre des mouvements de locomotion et de saut, en tenant compte de leurs propriétés respectives.

Finalement, nous considérons l'interaction d'un humain virtuel avec son environnement. En fonction de conditions initiales et finales imposées par la vitesse de locomotion et la position des pieds, nous proposons une méthode qui calcule la trajectoire correspondante. Pour illustrer cette méthode, nous étudions un cas reflétant au mieux le comportement d'un humain confronté à un obstacle : interactivement, à tout instant, des obstacles peuvent être créés face à un humain virtuel en déplacement. Notre méthode calcule en temps réel une trajectoire permettant à l'humain virtuel un franchissement précis de l'obstacle ainsi créé.

Acknowledgements

First of all, I would like to thank my thesis director, Professor Daniel Thalmann, who trustfully gave me the opportunity to work as a research assistant in his laboratory and to carry out my thesis. I have particularly appreciated the working conditions as well as the projects I participated in.

Many thanks to Ronan who has supervised my thesis work during these four years. He has always been available to answer my numerous questions, queries, doubts, doodles on the blackboard, even though he introduced an hourglass during the final phase of my thesis. I am still amazed by his patience and tolerance, severely tested in memorable parties at Satellite.

I also wish to thank the jury members, Dr. Kamiar Aminian, Dr. Xuguang Wang as well as Professor Thomas Vetter for the care, interest and comments they have brought to the examination of this thesis.

I have also appreciated the good atmosphere within the VRLab and the discussions between co-workers. My thanks notably go to Benoît Le Breton de Skifiex who has sometimes misused inverse kinematics, Etienne for the computer hardware support and my two flat screens, Mireille for the videos and her Humagne Blanche, Sébastien for his valuable help in programming, Anderson for singing with his toothbrush, Frédéric for his advice to buy a big car, Pablo for his lasting but essential theories, Helena for the careful reading of my english literary masterpieces. Thanks also to Josiane for her availability, her good mood and the organization of my travels.

I equally thank all the people who have accepted to act as guinea pigs, by wearing the comfortable stretch suit for the motion capture sessions. Lorna, Mireille, Raquel, Stéphanie, Ali, Benoit, Etienne, Joël, Sébastien and Vincent have sportingly run on a treadmill and jumped gracefully with small pumps.

I express my profound thankfulness to Pascal and Remo, fellow students from the ETHZ, who have read and commented this thesis with pertinence and objectivity. A special thought for my parents who gave me as a present the Pécub's cartoons which add a welcome humoristic style to this scientific work. Thank to the cartoonist for his collaboration, kindness and talent.

Finally, I would like to show all my gratitude to my friends for the good time passed together, and who have helped me escape from the virtual world. I would also like to very sincerely thank my parents Françoise and André as well as my family for their unconditional support and encouragements in my choices and projects, even during difficult times. Particularly, many thanks to Sonia for her unremitting presence, patience and attention.

Remerciements

Tout d'abord, j'aimerais remercier mon directeur de thèse, le Professeur Daniel Thalmann, de m'avoir offert un poste d'assistant dans son laboratoire et de m'avoir accordé sa confiance pour la réalisation de ma thèse. J'ai particulièrement apprécié les conditions de travail ainsi que les projets auxquels j'ai participé.

Un grand merci à Ronan qui a supervisé mon travail de thèse tout au long de ces quatre années. Mes nombreuses questions, interrogations, doutes, gribouillages au tableau ont toujours trouvé réponse dans son bureau, malgré le sablier qu'il utilisait durant la dernière phase de ma thèse. Sa patience et sa tolérance m'ont toujours étonné, mises à rude épreuve lors de mémorables soirées à Satellite.

J'adresse également mes remerciements aux membres du jury, le Docteur Kamiar Aminian, le Docteur Xuguang Wang ainsi que le Professeur Thomas Vetter pour le soin, l'intérêt et les commentaires apportés à l'examen de cette thèse.

J'ai également apprécié la bonne ambiance au sein du VRLab et les discussions entre collègues de travail. Mes remerciements vont notamment à Benoît Le Breton de Skifiex qui a parfois abusé de la cinématique inverse, à Etienne pour le support matériel et mes deux écrans plats, à Mireille pour les vidéos et l'Humagne Blanche, à Sébastien pour sa précieuse aide en matière de programmation, Anderson pour ses chansons avec brosse à dents, Frédéric pour ses conseils sur l'automobile, Pablo pour ses longues mais indispensables théories, Helena pour la lecture et la correction de ma prose anglaise. Un merci particulier à Josiane pour sa disponibilité, sa bonne humeur et l'organisation de mes escapades.

Je remercie aussi tous les cobayes qui ont bien voulu revêtir la confortable combinaison moulante de motion capture pour courir sur un tapis roulant et sauter avec des petites chaussures de danseurs : Lorna, Mireille, Raquel, Stéphanie, Ali, Benoit, Etienne, Joël, Sébastien et Vincent.

J'exprime ma vive reconnaissance à Pascal et Remo, compagnons de l'ETHZ, qui ont relu et commenté cette thèse avec pertinence et objectivité. Un clin d'œil à mes parents qui ont eu l'idée de m'offrir les illustrations de Pécub qui ajoutent à ce travail scientifique une note humoristique bienvenue. Merci à l'artiste pour sa collaboration, sa gentillesse et son talent.

Finalement, je témoigne toute ma gratitude à mes amis qui m'ont permis de passer de bons moments pour s'évader du monde informatique. J'aimerais aussi remercier très sincèrement mes parents Françoise et André ainsi que toute ma famille pour m'avoir toujours soutenu et encouragé dans mes choix et projets, même dans les moments difficiles. Un merci spécial à Sonia pour sa présence, sa patience et son écoute de tous les instants.

Contents

I Preambule 7

1 Introduction 9
- 1.1 Context 9
- 1.2 Motivations and Objectives 10
- 1.3 Approach 11
- 1.4 Organization of this Thesis 12

2 State of the Art in Character Animation 13
- 2.1 Motion Synthesis 13
  - 2.1.1 Hand-driven Methods 13
  - 2.1.2 Model-driven Methods 14
  - 2.1.3 Data-driven Methods 16
- 2.2 Method Comparison 21
- 2.3 Parameterized Motion 23
  - 2.3.1 Scattered Data Interpolation 23
  - 2.3.2 Statistical Methods 26
- 2.4 Motion Constraints 29
  - 2.4.1 Constraint Detection 29
  - 2.4.2 Constraint Enforcement 29
  - 2.4.3 Motion Anticipation 32
- 2.5 Motion Blending 32
- 2.6 Motion Planning 36

II Motion Modeling 39

1 Fundamentals 41
- 1.1 Data Acquisition 41

1.2 Principal Component Analysis 43
1.3 A PCA Algorithm for Motion Data 44
1.4 Conclusion 47

2 Generation of Motion Units 49
2.1 Introduction 49
2.2 PCA on Angular Data 50
2.3 Walking and Running Motion Units 53
  2.3.1 Single High-level Parameter Variation 53
  2.3.2 Multiple High-Level Parameter Variation 57
    2.3.2.1 Main PCA Level 57
    2.3.2.2 Sub-PCA Level 1 60
    2.3.2.3 Sub-PCA Level 2 62
    2.3.2.4 Motion Unit Computation 63
    2.3.2.5 Hierarchical Order 64
2.4 Jump Motion Units 64
2.5 Results 66
2.6 Conclusion 68

III Character Animation 71

1 Locomotion and Jump Engines 73
1.1 Introduction 73
1.2 Space Normalization 73
1.3 Time Normalization 74
  1.3.1 Walking and Running Frequencies 75
  1.3.2 Jump Frequency 77
1.4 Continuous Motion Generation 80
1.5 Results 82
  1.5.1 Validation 84
  1.5.2 Discussion 87
1.6 Conclusion 90

2 On-line Adaptive Footplant Detection and Enforcement 93
2.1 Introduction 93
2.2 Motion Anticipation 95
  2.2.1 Current Posture Computation 95
  2.2.2 Anticipated Posture Computation 95

2.3 Footplant Detection . . . 98
  2.3.1 Thresholds on Cartesian Position and Speed . . . 98
  2.3.2 Adaptive Positional Threshold . . . 99
  2.3.3 On-line Detection . . . 102
2.4 Footplant Enforcement . . . 102
  2.4.1 Inverse Kinematics Model . . . 103
  2.4.2 Ease-in with Anticipation . . . 104
  2.4.3 Ease-in Activation . . . 105
    2.4.3.1 Constant motion parameter . . . 106
    2.4.3.2 Planned motion parameter variation . . . 106
    2.4.3.3 Modification of a planned motion parameter variation . . . 106
2.5 Results . . . 107
  2.5.1 Putting it all together . . . 107
  2.5.2 Foot Sliding Correction . . . 108
  2.5.3 Stylistic Edition . . . 109
  2.5.4 Continuous Parameter Variation . . . 109
  2.5.5 Curved Path . . . 110
2.6 Conclusion . . . 111

3 Coherent Transition Generation between Locomotion and Jump **113**
3.1 Introduction . . . 113
3.2 Method Overview . . . 115
3.3 Controlled Blending . . . 117
  3.3.1 Coherent Motion Adaptation . . . 117
    3.3.1.1 Final Run-up Speed . . . 117
    3.3.1.2 Run-up Duration . . . 118
  3.3.2 Transition Time and Duration . . . 120
  3.3.3 Time-Aligned Blending . . . 121
3.4 Results and Conclusion . . . 124

IV Applications **127**

1 Obstacle Handling in Dynamic Environments **129**
1.1 Introduction . . . 129
1.2 Strategic Planning . . . 130
1.3 Path Determination . . . 132
  1.3.1 Getting Round . . . 133
  1.3.2 Jump Run-up Computation . . . 133

1.4 Animation 138
1.5 Results 139
1.6 Conclusion and Discussion 142

2 Software Implementation 145
2.1 PCA Hierarchical Structure 145
2.2 LibLOCO and LibMOCO Libraries 145
2.3 Applications 146
2.3.1 Scripted Motion Generation 146
2.3.2 On-line Motion Generation 147

V Final Considerations 151

1 Conclusion 153
1.1 Summary 153
1.2 Contributions 154
1.2.1 Motion Parameterization 154
1.2.2 Constraint Detection and Enforcement 155
1.2.3 Coherent Motion Transition 155
1.2.4 Dynamic Obstacle Handling 156
1.2.5 Integration and Applications 156
1.3 Perspectives 157
1.3.1 Result Improvements 157
1.3.2 Concrete Applications 157
1.3.3 Future Extensions 158

A List of Notations 159

B Background Mathematics 163
B.1 Basic Statistics 163
B.1.1 Mean 163
B.1.2 Variance 163
B.1.3 Standard Deviation 164
B.1.4 Covariance 164
B.1.5 Correlation 164
B.2 Matrix Algebra 165
B.2.1 Eigenvalue Decomposition 165
B.2.2 Singular Value Decomposition 165

B.2.3 PCA on Single Dataset 166
B.2.4 PCA on Sample Dataset 167

C Fit Functions on PCA Spaces **169**
C.1 Walking and Running Motion Units 169
C.2 Walking and Running Jump Motion Units 171

D Pecub's Cartoons **175**

Part I

Preambule

Chapter 1

Introduction

Computer Graphics encompasses techniques for the modeling and rendering of 3D scenes, as well as methods for the animation of these scenes. Computer Animation is deeply involved in this concept of bringing objects to life, by changing their state along time, and more specifically, it is engaged in making virtual humans move. This thesis is dedicated to the synthesis of human movement and aims at controlling the animation of virtual characters in their environment. The large motion diversity forces us to focus on two elementary human activities: locomotion and jump.

1.1 Context

For centuries, the study of movements perceptible in nature has been an interesting and captivating subject of research. From simple observations, such as a rolling stone, to more sophisticated ones, such as the flow of the water or the dispersal of a gas, many types of movements have been studied. One of the most fascinating movements concerns human motion, whose measurement and analysis would provide with a wide palette of perspectives.

The technology evolution has been the catalyst for human motion studies. Thanks to the invention of photography (discovered in 1839 by Daguerre), human activities can be decomposed in time and space by superimposing photographs taken at a given rate. At that time, the main interest was the understanding of the laws and models which are at the origin of the human motion generation. In parallel, the cartoon technique took its first steps in the animation of objects and characters by decomposing actions in successive painted images. Later, the arrival of the first computers allowed for the calculation of motion laws and the automation of animation processes.

At the very beginning of computer generated animations, it was however impossible to obtain complex characters. They were represented merely by single points which were moved in a 2D scene, like in the early arcade games. With the increase of computation performance, research interest and commercial investment, new virtual human models appeared. From simple 2D articulated systems, the characters became complex 3D structures. Concurrently, a vast range of animation techniques emerged as well. One of the first, which is known as keyframing, reproduces the concept of cartoons by manually specifying a sequence of character poses. However, this consumes huge labor time and requires artistic talent, which is prohibitive for most applications. Later, another issue was investigated. It consists in developing physical models, based for example on motion observations, like those provided by photography, or on motion simplifications. While this approach guarantees that resulting motions are physically correct, motions that are physically plausible may nonetheless appear unnatural, awkward or robotic.

In the middle of the eighties, a novel technique in the framework of human motion analysis and synthesis is launched: motion capture. This technique allows to record the performance of a human in 3D, and to reproduce it with high-fidelity onto a virtual character. The main markets which benefit from motion capture are medicine for gait analysis and rehabilitation, prosthetic design, sport for a complete and accurate performance analysis, video games (player first person, sport games), movies (digital crowds, stunts), television (TV shows, advertising, music videos), Virtual Reality with immersion (treatment of social phobia) and urban control (traffic simulation, crowd surveillance). Stimulated by those numerous application fields, motion capture has rapidly evolved towards non-invasive equipments with a large capture volume and real-time animation feedback onto a virtual character.

Unfortunately, motion capture is not the panacea and suffers from its lack of flexibility. As this technique can only be used to reproduce a recorded motion onto a character matching the performer's proportions, it offers little control over character's actions. Moreover, imagine that you want to animate a character alternately walking and running with a varying speed along a given path, and avoiding obstacles. It is clearly expensive to capture every desired sequence. Similarly, it is tedious and difficult to directly edit an original motion which nearly corresponds to your goal animation. Hence, the production of such animation sequences first requires a manual selection of appropriate motion clips over a vast motion capture database. It demands a relevant assembly of those clips and their parameterization. Existing methods allow an automation or semi-automation of this process but lack in intuitive motion parameterization, flexibility for motion assembly, performance for on-line motion synthesis, and/or reaction to dynamic environments. It is in this context that we propose this thesis.

1.2 Motivations and Objectives

Locomotion, such as walking or running, as well as jumping, are some of the most basic forms of daily human motions. In video games or character animations, it is therefore natural to observe scenes with virtual humans in locomotion and jumping over obstacles. The importance of locomotion is also revealed by commercial software packages, which include character animation engines in various forms. However, the development of such engines is still in progress on account of several constraints and difficulties.

The challenge consists in combining motion realism with a fine motion control while preserving on-line performance, a very important aspect in video games for example. Realism is quite difficult to obtain in human animation. On the one hand, the human body structure is extremely complicated. On the other hand, people are natural experts in human motion; inaccuracies in animated motion can readily be identified, even though they do not know exactly why. In addition, the character's interaction with its environment is not to neglect. Currently, the animation approaches which take into account the environment need to know the complete scene and its obstacles in advance.

The combination of these requirements into a complete on-line animation system, coupled with a coherent and reactive behavior of the animated character with respect to its *a priori* unknown environment, would greatly increase the autonomy of virtual humans for many application fields.

Motivated by such assertions, the aim of this thesis is to create an animation engine allowing for the on-line locomotion generation of a character capable of jumping over obstacles in a virtual environment. This engine must provide significant motion realism together with a high-level intuitive control of the motion parameters (like linear or angular speed) to simplify the animator's work. In addition, we emphasize on the need for a generic engine able to animate any character size. To test the on-line aptitude of our motion model, we also aim at developing a reactive method to adapt the locomotion according to objects appearing in the environment.

1.3 Approach

The ideal animation engine is the one that exactly matches the human motions, with its subtle and unpredictable changes and variations. Moreover, this ideal model is driven by the ability to perceive the environment, inducing motion adaptation.

Along the next chapters, we will see that the existing motion models in Computer Graphics are far from being ideal. In fact, they are focused on a specific aspect of the animation and neglect other requirements. For example, a method may produce excellent and accurate animation results, but only for off-line use. Another technique may animate a character with obstacle avoidance, however, without ensuring the continuity along the generated motion.

Our approach proposes the elaboration of a single system which encompasses real-time animation generation and environmental interaction. The motion generation and interaction models are carefully developed according to the main requirements for character animation:

- Based on a motion capture database, our model must produce high-fidelity results;
- The database dimension has to be reduced by applying a statistical algorithm to improve the efficiency of our methods and facilitate an intuitive high-level motion parameterization;
- Our model must be generic, firstly, in order to be used for locomotion actions as well as jumping, and secondly, in order to be applicable to any character size;
- Based on real observations, the locomotion and jumping actions must be automatically, coherently and smoothly assembled;
- The final motion quality can be enhanced by enforcing constraints to preserve foot interactions with the ground;

- Our model must be able to react to obstacles by adjusting the motion parameters automatically according to *a priori* unknown environmental configuration.

Such an approach must provide a real-time animation tool useful for the generation of realistic virtual human motion, controlled in an on-line manner either directly by animators or by a simulation process which distributes high-level tasks to characters. This animation tool should also take the environmental constraints into account through automatic motion modifications.

1.4 Organization of this Thesis

This thesis is structured into five parts composed of several chapters. The first part contains two chapters: one is this present introduction and the other reviews essential techniques in the form of a state of the art dedicated to Character Animation.

The second part introduces in its first chapter the fundamentals which are used for our animation model, including the description of our motion database construction, as well as a statistical method for data dimension reduction. The second chapter precisely describes our motion modeling to parameterize basic structures of locomotion and jumps with high-level intuitive parameters.

The third part focuses on real-time character animation and is divided into three chapters. In the first one, we explain the generic motion synthesis, for locomotion on the one hand, and for jumps on the other hand. The second chapter proposes a technique to detect and enforce foot constraints. Finally, a method for smoothly and coherently assembling locomotion and jumping activities is proposed in the last chapter.

Applications of our animation engine are discussed in the fourth part. Its first chapter presents a method for handling obstacles in a dynamic environment. The second chapter exposes software tools which have been implemented to ease the production of animations.

Finally, to conclude this thesis, the last part proposes a conclusion and a summary of the contributions, as well as several appendices, notably a list of the symbols used in this document.

Chapter 2

State of the Art in Character Animation

In this chapter, we present research related to the human motion synthesis. We first present a general overview of common techniques, and then we focus on a detailed review of previous work for each aspect covered in this thesis: parameterized motion, motion constraints, motion blending and motion planning (from Section 2.3 to 2.6).

2.1 Motion Synthesis

Existing motion synthesis techniques can be divided into three research directions: hand-driven, model-driven and data-driven methods. In this section, we present and discuss previous work according to these categories.

2.1.1 Hand-driven Methods

The hand-driven methods are the oldest and simplest ways for animation creation[Sturman, 1985]. An animator determines manually postures (”key-frames”) of an articulated character by defining positions and orientations on its joints at specific animation times (”key-times”). The final motion results in a smooth interpolation between the corresponding key-frames.

On the one hand, these methods are dependent on the animator technical and artistic skills. The choice of the key-frames is crucial to reproduce a motion as realistic as possible. In addition, the posture design is complex due to the important DOFs number a human skeleton is composed. On the other hand, commercial software [Maya®, 2005; MotionBuilder®, 2005; 3ds Max®, 2005; Carrara®, 2005] are available to help the animator (Fig. 2.1). They provide tools for the animation pre-visualization, interpolation methods or rigs to manipulate body parts easily. Even offering a great control on the motion, these methods are quite labor intensive as many postures must be created, in general 25 per animation second. As the invested time increases drastically for complex skeletons, hand-driven techniques are more adequate for cartoon animation. In this case, the used characters are based on simplified skeletons as the realism is less demanding.

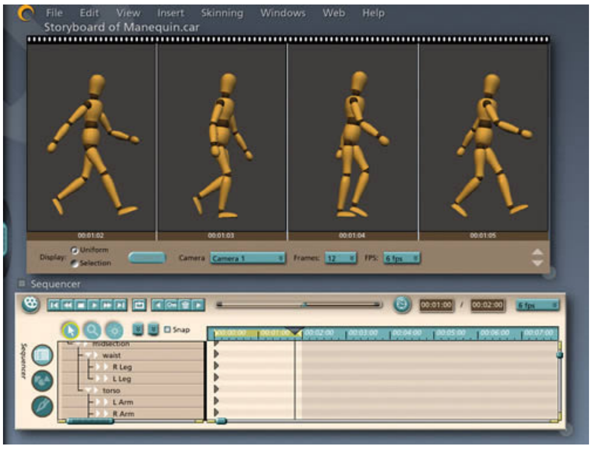


Figure 2.1: Key-frame creation in the commercial software Carrara ([Carrara®, 2005]).

2.1.2 Model-driven Methods

Model-driven methods explicitly describe how movements are generated, and use computation to propagate changes in high-level parameters to changes in low-level parameters. Therefore, these methods typically have a small number of high-level parameters which can be modified to change the generated motion. We can distinguish two classes of model-driven techniques, either based on kinematics or physics.

Kinematics Most of the kinematic approaches rely on the biomechanical knowledge, and combine direct and inverse kinematics [Multon et al., 1999]. They do not necessarily preserve physical laws and focus on the kinematics attached to a skeleton (positions, speeds and acceleration of the rigid body parts).

During the eighties, methods have been developed to generate locomotion patterns, driven by high-level parameters such as step length and frequency. Zeltzer in [1982; 1983] defines finite state machines to control the synthesized human gait. The states represent bundle of key-frames which are interpolated to produce a desired walking motion. Boulic et al. in [1990; 2004] presents a walking engine built from experimental data on a wide range of normalized speed. It allows to animate virtual humans of any size driven by two high-level parameters: the linear and angular speed.

Other approaches aim at ensuring that the feet do not penetrate into the ground. Bruderlin and Calvert [1989] introduce a method which changes the character's root so that its origin is always fixed on the support foot during the walking motion. As the leg in this system contains only one joint, the resulting animations may look artificial. The authors improve this technique by adding more DOFs to model the leg. It produces smooth and parameterized walking gaits [Bruderlin and Calvert, 1993]. Finally, an analogous method is applied to running motions [Bruderlin and Calvert, 1996]. In [Chung and Hahn, 1999], the foot position of the support leg is controlled prior to the swing leg. In addition, a collision avoidance module is used to generated stair climbing walking (Fig. 2.2, a).

More sophisticated methods offer other motion controls. The method presented in [Sun and Metaxas, 2001] adapts the walk to uneven terrain (Fig. 2.2, b) by deforming original motion generated in a 2D space of step length and step height parameters. The original motions are represented in the sagittal plane whose orientation is progressively modified according to the locomotion direction. Tsumara et al. [Tsumura et al., 2001] propose a locomotion well adapted for brisk direction changes, by handling the next footprint positions. However, the speed can not be controlled by the user.

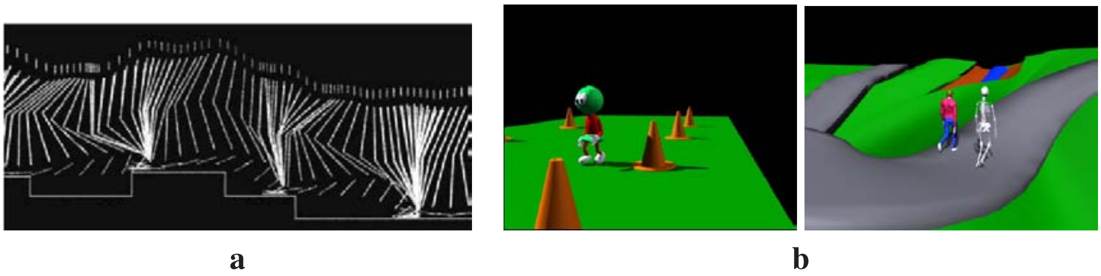


Figure 2.2: Example of walking motions obtained by kinematic approaches. **(a):** Stairs climbing [Chung and Hahn, 1999]. **(b):** Walking for curved path and uneven terrain [Sun and Metaxas, 2001].

One of the main drawbacks of these methods concerns their lack of motion realism. The animations look too robotic and seem not very familiar for the human eye. Therefore, physical models improve those negative aspects.

Physics Since real human movement is governed by the laws of physics, the elaboration of a physically-based model is a natural strategy for animating motion. Such a model computes each body part displacement as a function of their mass distribution and of the torques generated at each joint. The main difficulty of this approach consists in specifying of all forces to apply to a system so as to move it. While average mass distribution can be found in the biomechanics literature [Winter, 1990], the determination of joint torques to achieve a particular motion is difficult (i.e. the ”control” problem).

Earlier works [Girard and Maciejewski, 1985; Girard, 1987; Raibert and Hodgins, 1991] propose hybrid approaches by combining kinematics and dynamics to animate simple legged figures. However, these pioneering techniques are not able to animate the full body of a complex character. Controllers have been therefore used in order to provide the forces and torques to apply to the body parts, according to given constraints. Hodgins et al. [1995] propose control algorithms based on finite state machines to describe a particular motion (running, bicycling, vaulting), and on Proportional Derivative (PD) servos to compute the joint forces and torques. This method is improved by Wooten et al. [1996; 2000] allowing the switch from one to another controller for generating transitions between jump and landing motions, while preserving balance. In [Faloutsos et al., 2001b], the balance is also controlled to generate motions recovering from a fall (Fig. 2.3). Ko and Badler [1996] propose a method which first generates a locomotion sequence using kinematics. Then dynamics rules are enforced to improve the animation realism, notably by maintaining balance with respect to human strength limits.

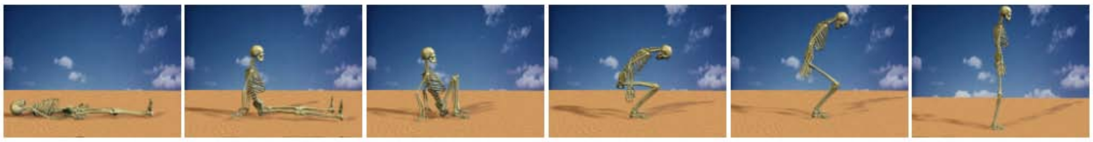


Figure 2.3: Motion sequence of a recovery from a fall [Faloutsos et al., 2001b].

The use of dynamics to control the motion of an articulated figure is very difficult. On the one hand, controlling an n DOFs body dynamically means controlling n actuators; this is at least as difficult as, and much less intuitive than, controlling n joints directly. On the other hand, once a controller has successfully been created, one can attempt to adapt it to new circumstances. For example, Hodgins and Pollard in [1997] introduce a technique for adapting physical controllers to different morphologies and skeleton types. For cyclic motion like walking, Laszlo et al. [1996] improve the motion stability in a dynamic environment by adding a closed-loop control to an open-loop system. The closed-loop control is based on forcing the gait motion back to a stable limit cycle after a perturbation is applied. The control is effected by manipulating hip pitch and roll. Finally, Faloutsos et al. [2001a; 2003] propose a method allowing the composition of different controllers in order to generate, still realistic, but more complex motions by using simpler controllers. The composition can be performed manually or determined automatically through learning methods like Support Vector Machine (SVM).

Besides being costly in computational time, dynamic simulation can generate physically plausible but not necessarily believable motions. Neff and Fiume [2002] propose creating more natural looking motion through an antagonist joint control model in place of a PD control. This system allows the modeling of tension and relaxation for specific joint, making a motion stiffer or lither. However, dynamic simulation often produces motion that lacks important features of natural human motions. Moreover, the controllers provide at present a relatively narrow range of realistic motions. Therefore, complex composite controllers have to be designed, capable of synthesizing a full spectrum of human-like motor behaviors. In addition, physically based methods demand too much user assistance with a high parameter dimension, inappropriate for human animation systems.

2.1.3 Data-driven Methods

Motion capture technique (Fig. 2.4) offers an alternative source of highly realistic movement sequences used for a variety of data-driven motion synthesis methods. The motion acquisition systems allow the estimation of the joint positions and/or orientations of a real performer. These parameters are then adapted to a virtual character in order to result in an animation which reproduces the original motion accurately. We distinguish five main categories of motion capture systems: video, optical, magnetic, mechanical and ultrasound systems. They all own their advantages and drawbacks influencing directly on the motion quality and time production [Menache, 2000]. The price of such systems is quite expensive and a strong practical experience is needed before using them. Recently, motion capture technique has been improved to use it at home as an interface for computer and video games. The idea consists in reducing the number of markers placed on the body's performer and the camera setup which is normally expensive [Chai and Hodgins, 2005]. A video-based method has been also proposed [Starck et al., 2005] to combine image based reconstruction and video-based animation techniques, allowing to capture motions with camera studio. It allows to provide animations with the appearance of complex surface dynamics such as cloth.

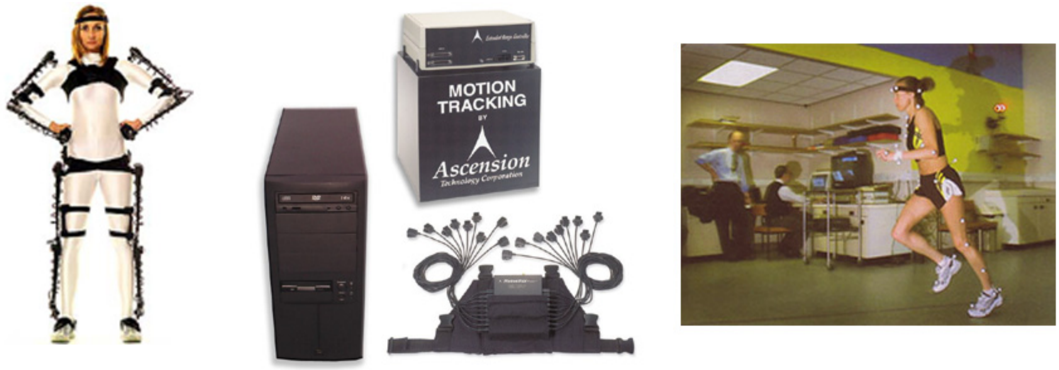


Figure 2.4: Motion capture systems. From **left** to **right:** Mechanical system ([Gypsy 4®, 2005]), Magnetic system ([MotionStar Wireless 2®, 2005]), Optical system ([Vicon Motion System®, 2005]).

The main disadvantage of motion capture resides in the lack of flexibility: the final motion can not be easily modified (the motion has to be often re-recorded) and it is only valid for characters having similar proportions as the performer. Therefore, motion editing algorithms have been developed in order to adapt, modify or deform original motions.

Even most of the approaches classified under this section are based on raw data obtained by motion capture, they can also be applied to data coming from keyframe animation, physical simulation or even another data-driven synthesis algorithm. In addition, we focus on methods that are not directly related to the following themes: parameterized motion, constraints detection and enforcement, and motion blending. Those topics, strongly connected to this thesis, will be exposed in further sections.

Signal Processing Specific works have tried to represent the motion in a non Euclidean space so as to benefit from other properties. Bruderlin and Williams [1995] applied techniques from image and signal processing domain to designing, modifying and adapting animated motions. The motion is considered as a time-varying signal which is decomposed by applying a multiresolution filter [Burt and Adelson, 1983]. The low frequencies contain general, gross motion patterns, whereas high frequencies contain details, subtleties and most of the motion noise. The frequency bands can be individually adjusted by varying their influence on the final motion (Fig. 2.5, a). This method is applied to multitarget motion interpolation, waveshaping and motion displacement mapping. In [Unuma et al., 1995], a motion representation based on Fourier series is used to compare and quantify a characteristic (tiredness, sadness, joy) between similar movements (Fig. 2.5, b). For example, the parameter "briskness" is obtained by the subtraction of a "normal" walk from a "brisk" walk expressed in the Fourier domain.

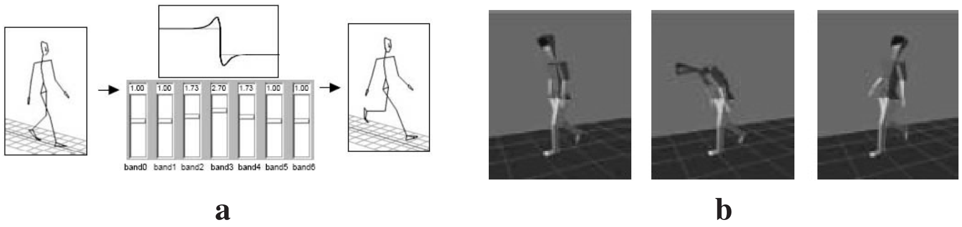


Figure 2.5: **(a):** Frequency band modification [Bruderlin and Williams, 1995]. **(b):** Interpolation and extrapolation of tired walk [Unuma et al., 1995].

In [Amaya et al., 1996], the difference, for a given subject, between an emotional and a neutral motion is expressed as an emotional transform. This latter is computed as a warp applied to the timing and amplitudes of the neutral motion. Finally, these warps can be applied to other neutral motion types in order to add similar emotional content. Perlin in [1995] uses principles from procedural texture synthesis to create subtle human movements such as shifting of weight and fidgeting while standing. Rythmic and stochastic noise functions determine time varying parameters that drive characters. The fundamental motion of each joint is sinusoidal with additive randomized noise used to prevent the motion from becoming repetitive. However, controlling the randomization is far from straightforward and may yield unpredictable results that can be physically impossible.

Motion Warping To deform a motion with precise goals, the first solution consists in modifying the body joint orientation in order to get a new posture. In [Witkin and Popović, 1995], the authors introduce a variant of displacement mapping called motion warping. The animator interactively defines a set of key-frames inducing a set of constraints. These are used to derive a smooth deformation preserving the fine structure of the original motion. However, it is difficult to ensure the geometric constraint enforcement between key-frames. In addition, motion warping methods are purely geometric techniques and operate on each DOF independently, without understanding the motion structure. They are not well suited for adjustments requiring coordinate movements, such as grasping actions with modification of object location. In such cases, not only the joint hand is affected, but also other joints like the elbow or shoulder.

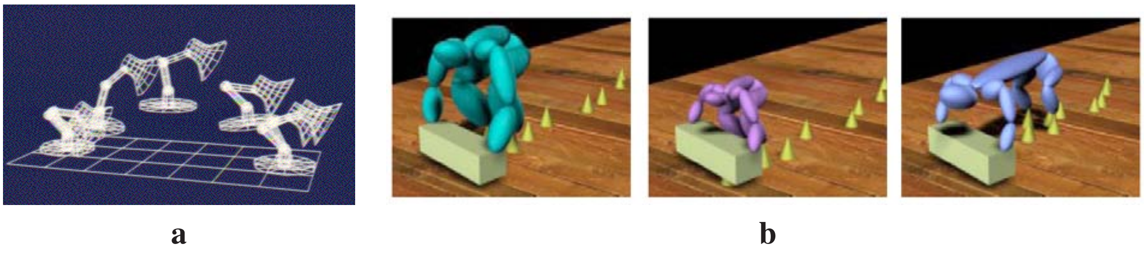


Figure 2.6: **(a):** The jumping luxo lamp [Witkin and Kass, 1988]. **(b):** Differently sized characters pick up an object [Gleicher, 1998].

Space-time Constraints (Kinematic) In order to alter coherently multiple DOFs of an original motion and over a continuous time period, constraint-based techniques can be applied. Discussed and classified in [Gleicher, 2001a], these methods provide effective tools to interactively manipulate a motion clip by changing some important movement properties. Space-time constraints were first introduced to the graphics community by Witkin and Kass [1988]. The authors demonstrate the viability of this approach with a jumping Luxo lamp: its motion is quite compelling as it crouches in anticipation of a jump and compressed to absorb the impact (see Fig. 2.6, a). The user specifies the start pose, end pose, and a physically based objective function. Then the optimizer aims at finding the best motion satisfying the described constraints. This is performed by minimizing an objective function described with penalty functions and defined over animation time.

However, several factors limit the application scope of these space-time approaches. First, it is difficult to formulate a desired deformation as mathematical constraints. Secondly, the complexity of the system to optimize (local minima, computation time) forces to limit the posture number as well as the skeleton's DOF number. Thirdly, the animator has to specify carefully the time interval for which the motion is modified. In fact, a too big interval may lead to an important possible solution set inducing a convergence towards an undesired local minimum. On the contrary, a too short time interval may restrict the solution space so as to find any appropriate solution. Finally, this technique is not applicable to real-time animation as it needs the original motion over a time interval.

Subsequent research has focused on ways to make these constraint-based approaches more viable. Cohen in [1992] proposes an interactive motion deformation tool improving the constraint definition. The creation of space-time windows helps to subdivide the problem (and the complexity) into subset problems. In addition, the constraints can be expressed as conditional equations in order to activate constraints only under specific conditions (e.g. when the distance between two joints is smaller than a given threshold). The approach presented in [Gleicher, 1997] enables the user to interactively position characters using direct manipulation. The numerical constraint problem is then solved fast enough to provide interactive feedback. A similar technique is used in [Gleicher, 1998] to perform motion retargeting, i.e. to adapt an animation sequence from one character to another with different limb lengths and proportions (see Fig. 2.6, b). Also Gleicher [2001b] exploits a space-time constraint solver for interactively editing the path traversed by a character.

Space-time Constraints (Dynamic) Using those previous editing methods, it is very difficult to specify stylistic attributes such as ”gracefully” or ”like Charlie Chaplin”. Moreover, the physics is omitted which can lead to unrealistic situations. Instead of adjusting only kinematic properties, other algorithms have been designed to preserve physical correctness. In [Popović and Witkin, 1999], dynamics is introduced with space-time constraints defined on a simplified model. Hence, the original motion is projected onto a less complex skeleton, with fewer DOFs. The constraint system is then solved and the result is adapted to the original skeleton. The produced animations are dynamically not absolutely perfect, but are visually convincing. More recently, Liu and Popović [2002] apply also a simplified dynamical model and space-time constraint to generate animation from a small set of keyframes. This input data is firstly analyzed by extracting geometrical constraints. Then other basic physical laws are added in order to formulate a complete space-time system. Even if these methods produce pleasant animation, they are well adapted only for high-energy motions. In addition, the results are still dependent on the animator choices, like the simplified skeleton [Popović and Witkin, 1999] or the key-frames [Liu and Popović, 2002]. Fang and Pollard [2003] formulate the physical constraints on aggregate force which is a representation of all external forces and torques that would have to be applied to the character's root to explain its motion. The optimization is performed in time linear to the number of character DOFs, whereas in general derivative computation is of quadratic complexity.

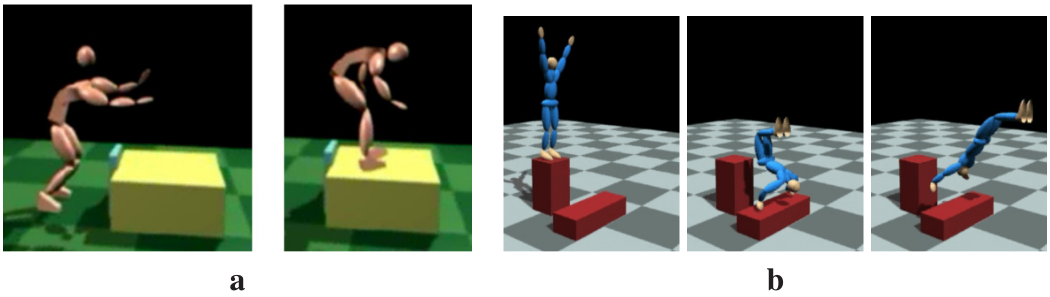


Figure 2.7: (a): Modified jump by raising the landing position and introducing an hurdle which forces raising of the legs during the flight stage [Popović and Witkin, 1999]. **(b):** An advanced handspring motion on an uneven terrain [Liu and Popović, 2002].

Instead of performing the optimization over the entire motion duration, Tak et al. [2002] propose to find the optimum value at every frame in order to run the method at interactive speed. Two consecutive filters are applied: the first enforces dynamic constraints per-frame and the second ensures the inter-frame consistency. To create more realistic motions through constraints optimization, Safonova et al. [2004] apply PCA (Principal Component Analysis) to selected motion capture data containing similar behavior than the required motion. Then, constraints expressed in the world frame (start/end postures, foot contact timing) are projected onto the low-dimensional space obtained by PCA. The optimization is solved in this space improving the efficiency and convergence thanks to the example motions. However, the final animation still does not approach the fidelity of captured motion.

Dynamic Simulation Motion capture is often combined with dynamic simulation for the purpose of producing realistic human like motion. In [Pollard, 1999], motion data is "scaled" in order to modify some parameters like character's features (e.g. the size or skeleton), the velocity and acceleration. The algorithm first fits a simple task model to the input motion. This task model, described with physics, is then scaled in a physically correct manner. The final motion is obtained by adjusting the scaled motion by the simple model to the roughly scaled input motion. Shin et al. [2003] introduce an efficient method to improve physical plausibility of motion capture data. Instead of using optimization techniques to find the more natural postures, this approach is based on hierarchical displacement map. The zero moment point (ZMP) constraint is enforced for ground phases (Fig. 2.8, a) and parabolic center of mass (COM) trajectory during flight phases. The simplicity and speed of this method is intended only for slight adjustments of an original motion.

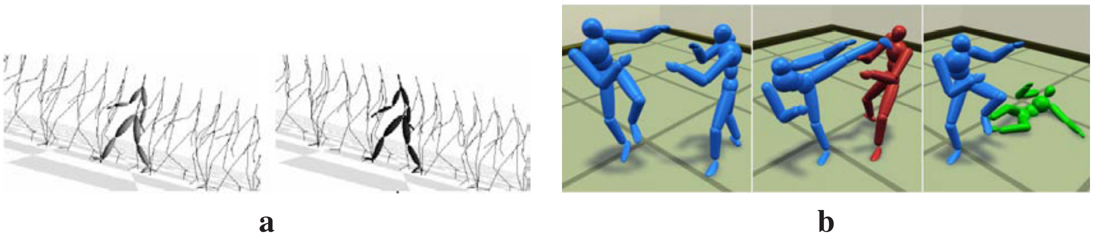


Figure 2.8: **(a):** Original and touched up climbing uphill [Shin et al., 2003]. **(b):** Simulated reaction of a kick (red) and reaction segment from a database (green) [Zordan et al., 2005].

Zordan and Hodgins [1999] present a system to capture the subtle details that make motion appear natural while maintaining physical realism. Upper-body motion data is converted to joint angles and used as the desired values for dynamic controller. This latter calculates using a proportional-derivative servo the appropriate torques so as to reach the desired angle values. In addition, environmental and task constraints are added to the system. This method is improved in [Zordan and Hodgins, 2002] by considering the whole human body with balance, control for hitting and simulated reactions to hits. For example, two characters can interact with boxing movements. Motion capture sequences are modified to hit specific locations, and the method simulates the corresponding reaction. The problem with this method is that once the effects of an impact are over, there are no schemes to return to motion capture control. For example, a punch can be as strong as to make the opponent falling down. Hence, Zordan et al. [2005] introduce a method which compares the simulated reaction with reaction segments from a motion library and determines the best one. The simulation is then smoothly blended with this best motion segments in order to generate a coherent and complete animation (Fig. 2.8, b).

2.2 Method Comparison

All presented methods, classified by hand-driven, model-driven and data-driven techniques have advantages and disadvantages. Illustrated in Fig. 2.9, the different approaches are summarized according to two criteria: the motion realism and the motion control freedom provided to an animator.

Hand-driven techniques allow the creation of realistic animations with a very important freedom for the animator to control the motion, constrained neither by visual realism nor physical accuracy. Actually, most of the 3D movies and games use keyframe animation sequences. However, this very flexible method has a price. It requires incredible investment of time and only skilled and talented designers produce realistic human motions. Model-driven approaches generate motions with less motion control freedom and realism. However, kinematic methods concentrate the control on a small set of high-level parameters, allowing an easier and more intuitive motion parameterization. In addition, the motion is generally created on-the-fly, but suffers from too robotic and jerky movements. The other part of model-driven methods, namely those based on physics, produce more realistic results but need lot of computational time (over several minutes). Moreover, for a valid resulting motion, physical accuracy does not imply visual realism. For example, a grasping movement can be simulated with various physically correct alternatives some of which may appear oddly and unnatural. Apart from these drawbacks, the motion control is driven by many parameters, difficult to directly interpret. Hence, data-driven methods have been developed.

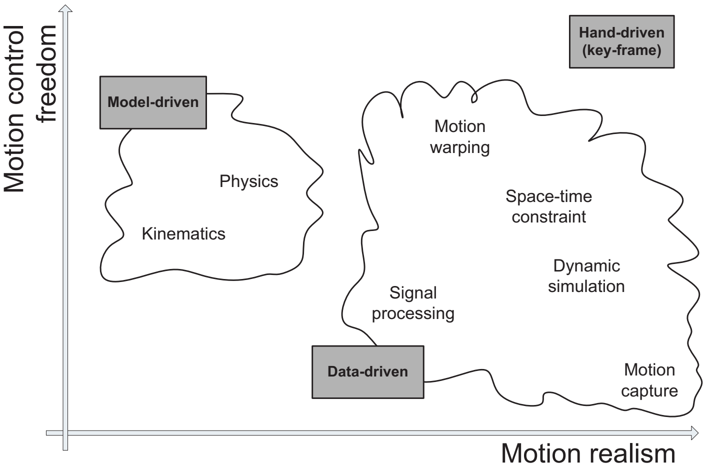


Figure 2.9: Summary of motion synthesis approaches according to the motion realism versus motion control freedom for animators.

Since the advent of motion capture systems which return high realistic motion without any control, the number of data-driven methods has exploded. Generally, these approaches produce motions with high realism thanks to the captured input data. The methods based on signal processing offer a weak control on the animation. In fact, the correspondences between motion parameters and frequencies are difficult to establish and the induced filtering can lead to less impressive results as the original motions. Motion warping techniques give lot of control freedom to the animators as key-frames are modified by hand. However, the motion realism is not ensured on the entire final sequences. Results can be improved with space-time constraint methods which consider a motion as a whole continuous sequence. In contrast, the constraint definitions have to be as intuitive as possible, reducing also the animator control. Finally, dynamic simulation allows to modify and enhance an original motion capture data by adding physical accuracy. The main advantage is that simple input data like rough key-frames are sufficient in order to produce convincing results. However, besides their expensive computational cost, these methods are limited to high dynamic motions. Nevertheless, the data acquisition is one of the stumbling blocks for data-driven methods. In fact, motion capture systems are expensive and need lots of pre-processing work to clean the recorded motions before using them. Moreover, while data-driven synthesis could be applied to any creature whose movements can be captured, in practice it is primarily applicable to humans. In contrast, most of the hand- and model-driven methods can be applied to a wider range of body, with varying topology and joint structure.

In the following sections, we focus on previous work strongly related to the topics covered throughout this thesis.

2.3 Parameterized Motion

This section is dedicated to the related work on the continuous parameterized motion generation, performed on-the-fly. This is the first main topic of this thesis. The focus is on data-driven methods based on motion capture data, producing currently results with the most realism and believability. The main difficulty concerns the elaboration of a parameterization which maps high-level motion characteristics with non-intuitive parameters, specific to the motion generation method. In this section, we discard approaches based on physics as they are too costly for real-time methods.

2.3.1 Scattered Data Interpolation

Guo and Robergé [1996] present one of the pioneering work which consists in a motion control mechanism driven by measurable terms like the velocity or step length. Their method converts a motion sequence composed of key-frames into an abstract parameterized curve referred to as a 1D frame space. These 1D frames are used to build higher-dimensional frame spaces where new sequences can be generated by linear combinations. The authors propose a frame space for locomotion driven by three parameters: the locomotion type, stride length and stride height. This method is improved in [Golam and Wong, 2000] by considering a motion with densely spaced signals instead of a small amount of key-frames. It allows to establish motion correspondences dynamically and produces better results for mixing stylistic different walking patterns. However, these two approaches are limited to a small number of input data in order to keep real-time performances.

Other approaches allow a multidimensional motion interpolation over a wide range of scattered input data. Rose et al. [1998] chose an interpolation scheme based on RBF (Radial Basis Function) to produce parameterized motions. Input motions are firstly manually classified by activities (”verbs”) and characterized by a parameter vector. The motion data is then represented by B-Spline control points which model the DOF functions over time. In addition, the motions attached to a given verb are structurally aligned by using a time warping process based on [Bruderlin and Williams, 1995]. To generate a new motion, a combination of RBF and polynomials is selected and provides the B-Spline coefficients corresponding to the requested parameterized motion. The polynomial function provides an overall approximation of the example motion space, while the RBF locally adjust the polynomial so as to get the exact example motion when the user gives its corresponding parameter vector (Fig. 2.10). Sloan et al. [2001] adopt cardinal basis functions for further performance improvements.

This RBF-based technique is incorporated in [Park et al., 2002a] for on-line locomotion synthesis and provide weights to be assigned to the example input motions. New motions are then generated by performing a weighted linear combination of the example data, using a multiple quaternion interpolation scheme. In addition and compared to previous approaches, this work proposes an on-line retargeting method based on [Shin et al., 2001] which allows to adapt the generated motion to different human size and terrain (Fig. 2.11, a). Recently, Mukai and Kuriyama [2005] improve the RBF function construction described in [Rose et al., 1998] by defining a specific kernel function for each input motion according to its characteristics. This approach is based on geostatistics which takes into account the correlation between

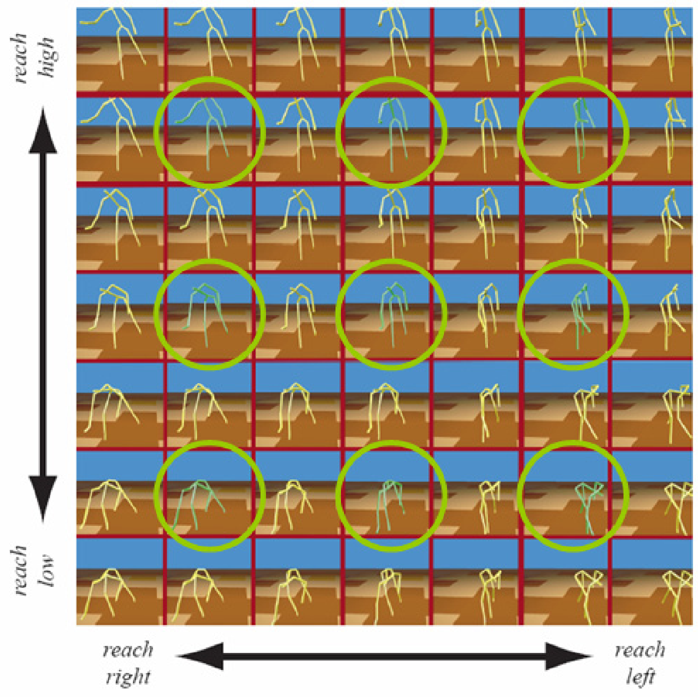


Figure 2.10: A reach sampled across two axes of the goal position for the hand. The green figures (encircled) are the example motions. The rest are created through the Verb and Adverb mechanism [Rose et al., 1998].

spatial distances and corresponding control parameters. It results in an accurate motion interpolation.

Even efficient enough to perform real-time motion synthesis, the above scattered data interpolation approaches force each input motion to contribute to the generate motion. On the one hand, it induces a computational efficiency which is dependent on the number of example motions. On the other hand, the method provides rough results when the user requests parameters far from the examples as interpolation weights are based purely on the linear approximation.

Therefore, other methods propose to parameterize motions with fewer examples. Pettré et al. [2002] propose to represent motion captured data into the frequency domain, similar to [Unuma et al., 1995]. The authors use walking cycles characterized with two parameters: linear and angular speeds. The original motions, expressed with Fourier coefficients, are projected into this 2D parameter space, and a Delaunay triangulation is performed. This approach is analogous to the one described in [Sun and Metaxas, 2001]. Hence, according to a given parameter pair, the three nearest neighboring motions can be selected to performed a weighted linear interpolation between them. However, the method may produce discontinuities when given new parameters a changeover from one to another triangle is necessary. In addition, the approach does not take into account the time-warping in order to align identical motion structures together. In [Kovar and Gleicher, 2003], a more general motion interpolation method is proposed, by addressing the time-warping among other problems. A new data-structure, referred to as registration curve, is introduced. This concept ensures automatically a consistent time-warping and an alignment of the humanoid root node for all input motions. In addition, physical constraints of the input motions are appropriately interpolated. To obtain a new motion, the user sets weights on manually selected motion examples (Fig. 2.11, b). Their attached registration curve allows to perform a consistent interpolation, based on the technique explained in [Park et al., 2002a]. Nevertheless, this method does not provide a direct correspondence between the weights and high-level parameter of motions and is therefore not intuitive for the animators.

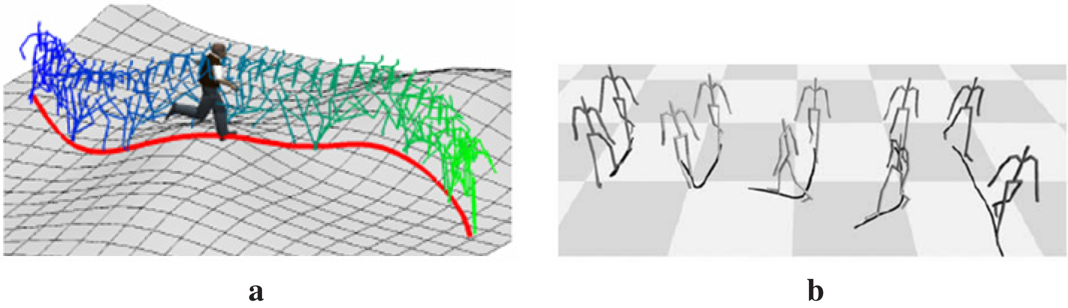


Figure 2.11: **(a):** Locomotion on a terrain [Park et al., 2002a]. **(b):** A parameterized clip build from a straight walk (far right) and one with a sharp 180 degree turn (fat left) is used to create sharp path changes in a continuum of directions [Kovar and Gleicher, 2003].

For motion interpolation, the selection of the necessary example motions over the entire input dataset can be performed automatically. The general strategy of sampling the space of interpolation is originally introduced by Wiley and Hahn [1997], leading to grids of regular samplings in the parameter space. According to a given parameter combination, a region can be determined in the parameter space and the interpolation is performed between the motions only included in this area. Other works are based on this strategy. Zordan and Hodgins [Zordan and Hodgins, 2002] generate dense sets of example motions as an aid for inverse kinematics tasks, while Rose et al. [Rose et al., 2001] improve the accuracy of the resulting motions by adding additional samples to parameter space.

Recently, another alternative method is proposed in [Kovar and Gleicher, 2004] to select the example motions. Given a segment of the motion data set (”query”), the method locates and extracts motion segments that are similar, representing the same action or sequence of action. This search is performed repeatedly, by taking the extracted segments as new queries. This process finished, the extracted segments are applied to perform a k-nearest-neighbors interpolation, as suggested in [Allen et al., 2002]. This allows to explicitly constraint interpolation weights to reasonable values and to project points outside the accessible region of parameter space back onto it. Lately, many works have been dedicated to search algorithm from motion databases. Forbes and Fiume [2005] propose to project the database and the query into a weighted PCA space. The weight assigned to the body joints can be modified according to the nature of the query. Another approach [Müller et al., 2005] alleviates the query formulation by the introduction of Boolean features like ”the right foot is in front of the left foot”.

In general, the weights assigned to example motions have no simple relationship to motion features. In [Kovar and Gleicher, 2003], the user has to set manually the weights and no motion extrapolation is possible. The methods based on scatter data interpolation [Rose et al., 1998; Park et al., 2002a] produce rough results when the user parameters are far from original input data as the interpolation is purely based on the linear approximation. Parallel to our research, Kovar and Gleicher [2004] have proposed a function mapping weights to a parameter vector. However, this approach assumes that the parameter are composed of joint positions and orientations, appropriate to generate goal-actions such as grasping or kicking, but not for movements where high-level parameters are required, like the locomotion speed for example.

2.3.2 Statistical Methods

Statistical methods are also applied to the motion parameterization problem. The method presented in [Pullen and Bregler, 2000] synthesizes 2D motion by adding random variations on the original training data. Correlations among numerous features of the data are taking into account and are modeled with a Kernel-based density representation of the joint probability distributions. This density representation is preferred as the shape of the distribution points is not a Gaussian pattern. Such correlation observations are also applied in [Pullen and Bregler, 2002] in order to enhance a sketched key-frame animation with motion capture data. Markov model is another technique used for 3D animation. In [Chenney and Forsyth, 2000], a Markov chain Monte Carlo algorithm is used to sample multiple animations that satisfy constraints for the case of multi-body collision of inanimate objects. For articulated character animation, Brand and Hertzmann [2000] uses Hidden Markov Models along with an entropy minimization procedure to learn and synthesize motions with particular style. Their appealing method computes automatically structural correspondences and extracts style between motion sequences (Fig. 2.12). This approach, even impressive, suffers form a complex mathematical framework which is dependent on a specific parameterization. In addition, the animations are not generated in real-time.

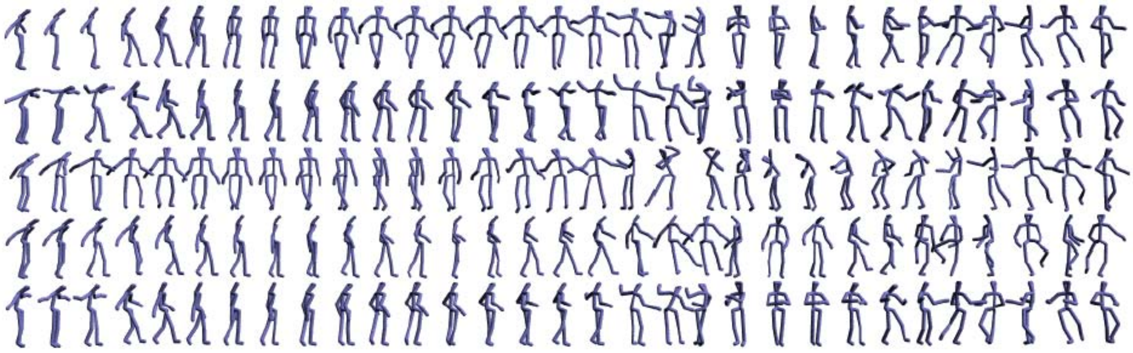


Figure 2.12: Five motion sequences synthesized from the same choreography, but in different styles (one per row). The actions, aligned vertically, are tiptoeing, turning, kicking, and spinning. The odd body geometry reflects marker placements in the training motion-capture [Brand and Hertzmann, 2000].

Principal Component Analysis (PCA) is another statistical method used in many and different fields for decades. This method is employed as a data compression technique to identify the significant variations in the data and eliminate the redundancy in the representation. Recently, PCA have been applied to Computer Graphics topics. The Morphable Model of 3D faces presented in [Blanz and Vetter, 1999] performs a PCA to estimate the probability distributions of a database of faces around their average. In addition, the initial database dimension is considerably reduced, improving computational performance. This model allows the parameterization of 3D faces according to facial expressions and attributes. Based on this system, it is possible to reanimate faces in images and video with different facial expression [Blanz et al., 2003] or to exchange faces in images [Blanz et al., 2004], independently of the illumination or viewpoint of the face to modify.

PCA is also well appropriate for data compression in animation. Alexa and Müller [2000] apply PCA to represent geometric key-frame animations allowing an adaptive compression. In [Bregler et al., 2002], a similar approach is used to capture the motion style of cartoons and retarget it into 3D models, 2D drawings or photographs (Fig. 2.13). For specific human animation, motion capture sequences represent often large data due to the high sampling rate and the important number of DOFs a virtual human body contains. A special attention has to be drawn for the motion data representation applied to PCA. When data are 3D joint positions, velocities or accelerations, PCA can be directly applied, as used in [Arikan et al., 2003] to reduce the dimension of motion feature vectors.

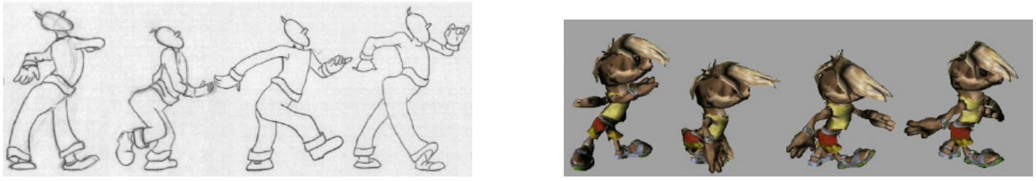


Figure 2.13: 3D example of key-shapes for the input cartoon and corresponding output key-shapes [Bregler et al., 2002].

For joint angle measurements which have a non-Euclidean geometry, it is necessary to approximate them into the Euclidean space by the use of the exponential maps for example [Pennec and Thirion, 1997; Grassia, 1998; Alexa, 2002]. In [Lim and Thalmann, 2002], PCA is firstly used to compress these data and then motion parameterization is performed on the reduced space using the RBF interpolation technique from [Rose et al., 1998]. The main interest of this approach is that each motion example is considering as a point in the PCA space, on the contrary to the following approaches. Tanco and Hilton [Tanco and Hilton, 2000] propose a motion synthesis system which uses PCA to perform a two level process based on Markov chain. However, the method is limited to a very small number of data (about approximately 500 frames).

The synthesis of idle motions in [Egges et al., 2004] is also based on PCA to facilitate operations such as blending and fitting of motions. This allows the production of small posture variations and personalized change of balance. Safonova et al. [2004] reduce the input data dimension using PCA so as to perform physically based motion synthesis more efficiently (Fig. 2.14). Troje [2002] presents an approach to parameterized walking motions, represented by 3D marker positions, with attribute like gender or mood. The method consists in applying firstly a PCA to each captured data and then to represent it by temporal sine functions. Finally, a second PCA is applied to all of these dimensionally reduced motions producing a new space where discriminant functions [Zhao et al., 1998] are used for determining the contribution of each dimension with respect to attributes. Despite of interesting results, this approach is limited for motion synthesis application. Firstly, the data are not represented into the joint angles spaces, inducing length modification for the body limbs. Secondly, an attribute change implies undesired consequences like the locomotion speed modification. Recently, a method [Grochow et al., 2004] based on Scaled Gaussian Process Latent Variable Model (SGPLVM) allows the mapping from a low-dimensional space (latent space) to a feature space which characterizes motions (joint angles, velocities and accelerations). Hence, a kernel function maps the correlation between postures according to their corresponding representations in the latent space. The method generalizes RBF interpolation, providing an automatic learning of all RBF parameters. The authors present applications for interactive character posing, which can replace conventional Inverse Kinematics solver. However, the SGPLVM technique requires some tuning and optimizations for its use in real-time motion synthesis based on large motion capture databases.

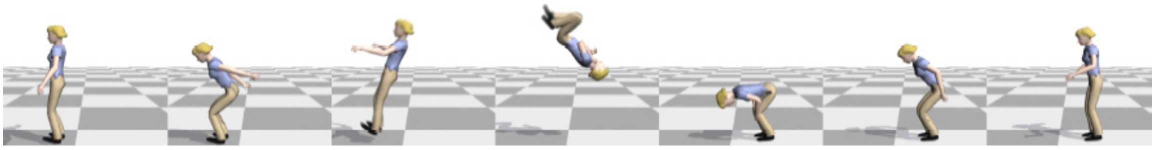


Figure 2.14: A back flip sequence synthesized using the method described in [Safonova et al., 2004].

Finally, another application domain use PCA for gait analysis, tracking and recognition. In [Gonzàlez et al., 2005], the authors suggest a comparison framework which allows to evaluate the variation of the joint angles between different subjects, while performing a same action like walking. The method differentiates between male and female walker. Concerning body tracking, Urtasun and Fua [2004a] use our PCA approach to reconstruct the 3D animation of a tracked human locomotion. This method allows also to recover motion parameters used for motion recognition [Urtasun and Fua, 2004b].

To summarize, previous works discussed above suffer from a number of limitations. First, there is no intuitive way to create a motion with specific high-level parameters, especially for motion generation beyond the captured domain (i.e extrapolation). To alleviate this problem, the input motions have to be separated into clusters, according to their parameters. Hence, the motion interpolation and extrapolation are precisely performed on those separated clusters, with intuitive and quantitative high-level parameters, like locomotion speed. Another limitation of the previous work concerns the motion adaptation to any kind of virtual character. In this thesis, we address to model a framework allowing the production of generic animations, applicable for various character sizes.

2.4 Motion Constraints

One of the main problems for parameterized motion techniques concerns the preservation of hard constraints. In this section, a hard constraint is considered from its geometrical point of view: for a given body part called end-effector, a goal position has to be reached and enforced within a period of time. A well known example of such constraint exists in locomotion patterns where the foot has to be fixed for a given duration when it touches the ground (referred to as footplant). However, a constraint has to be detected firstly, by identifying over time its begin and end, and then to be enforced over this duration at a given position.

2.4.1 Constraint Detection

Very few results on constraint detection can be found in the literature. A specific category of methods allows the detection of footplants in motions. Several methods [Kovar et al., 2002a; Menardais et al., 2004] apply user-chosen threshold values to detect those constraints. A footplant is identified when both position and velocity of the foot are under theses threshold values. Lee et al. [2002] extend this approach for constraint detection between body segments and objects in a environment. The method considers relative position and velocity between a segment and an object in order to decide whether they are in contact or not.

Bindiganavale and Badler [1998] introduced a method based on the concept of the effector acceleration zero-crossing, allowing to avoid checking constraint at every animation frame. The collision between two points of interest, a virtual human's finger and a table for example, is checked only at zero-crossing of the second derivative of these points. Hreljac and Marshall [2000] use a similar technique to determine the heel-strike and toe-off times in walking motion. Their results are compared with measures performed on force platforms.

Finally, Liu and Popović [2002] present a generic method to detect constraints. All the points on a character's body remaining fixed for some period of time are identified as constrained. In addition, close constraints are merged if the duration between constraints is under specified threshold values. Salvati et al. [Salvati et al., 2004] extend this method to detect constraints relative to moving objects in a scene (e.g. a hand touching a ball).

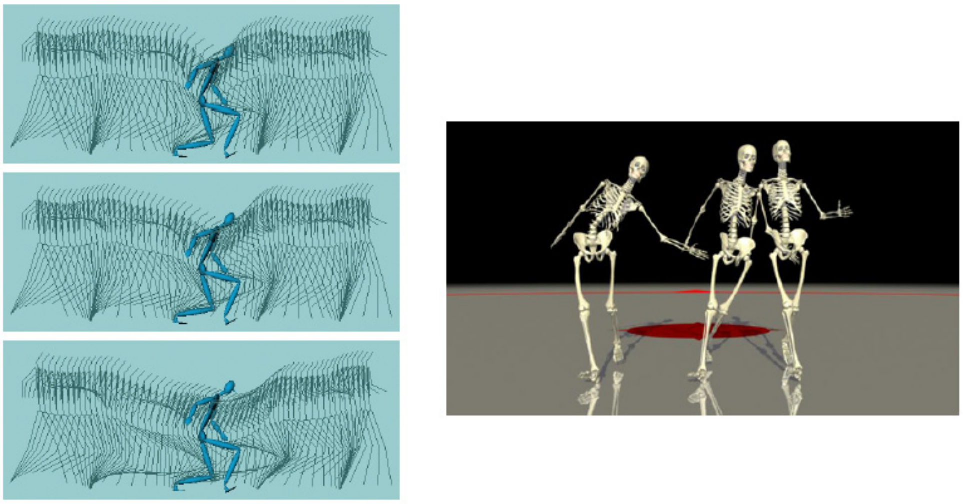
In addition to being off-line, all those techniques prove to be unreliable if the original motion is noisy. In fact, the threshold values are closely related to the nature of the considered motions. As a consequence, they may vary whether the motion is a walking, running or jumping activity, inducing a manual fine tuning of these threshold values. In this thesis, we propose a footplant detection method which is on-line and determines threshold values automatically with respect to the motion characteristics.

2.4.2 Constraint Enforcement

Once a constraint is detected, it has to be enforced in order to adjust its associated end-effector at a specified positions and/or orientation. Constraint-based techniques discussed in Sub-section 2.1.3 (paragraphs Space-time Constraints) are computationally expensive and remain difficult to provide a correct mathematical formulation corresponding to the desired enforced constraints. Hence, this enforcement stage can be solved by applying the Inverse Kinematic (IK) technique. Its use in animation dates back to some of the earliest system [Girard and Maciejewski, 1985; Korein, 1985]. Roughly explained, the IK algorithm automatically computes value of each individual DOF of an articulated system in order to satisfy a given task usually expressed in Cartesian space. This technique requires the resolution of complex non-linear equations and is usually expressed as a constraint-satisfaction problem.

Two main classes of IK algorithms can be distinguished: analytical and numerical. For very simple robotic manipulators with few DOFs, analytical (or closed-form) solutions can be found by direct resolution of the non-linear equations [Paul, 1981]. The advantages of those approaches are multiple. They are based on a fast computing, ideal for real-time application, and on robust solutions ensuring smooth resulting trajectories without ill-conditioned equations in certain character configuration [Maciejewski, 1990]. In the character animation field, several researchers have addressed the case of arm and leg control: Korein [1985] provides an interesting analytic solution based on explicit joint redundancy and applied to a seven DOF arm that deals with joint limits, and Tolani et al. [Tolani et al., 2000] also discuss similar procedures. Inspired from these approaches, Lee and Shin [1999] propose a framework for motion editing based on a simplified IK algorithm. The user-defined constraints which must modify an original motion are satisfied with a hierarchical sequence of adjustments. At each stage, the IK algorithm is performed independently on each frame and a spline is fit to the resulting displacements, with the knot spacing growing smaller at later stages of the algorithm (Fig. 2.15, a). Another simplified analytical IK algorithm [Shin et al., 2001] allows on-line skeletal reconstruction from motion capture data with a particular attention for the interaction between end-effectors and objects in the environment by measuring their proximity. Finally and closer to our intention, Kovar et al. [Kovar et al., 2002b] introduce a specialized analytical method to clean-up footskate from motion capture data. The authors take measures to avoid ”popping” artifacts that can occur when a limb is near full extension. In such situations, the key idea consists in extending the leg length. However, this ”trick” is not usable for an animator who works with virtual characters as similar as possible to real humans and therefore represented by a rigid body like those based on H-ANIM [2005]. In addition, this footplant enforcement method needs to delay animation frames. Consequently, this solution not intended for on-line motion synthesis.

However, despite its intrinsic efficiency, analytic IK suffers mostly from lack of flexibility. Numeric IK (see [Welman, 1993] for a survey) address this issue by solving sophisticated constraints for an arbitrary number of end-effectors and joints on complex skeletons [Zhao and Badler, 1994]. For instance, Yamane and Nakamura [2003] divide the constraints into two priority layers. Weighted priorities can also be assigned to the tasks so as to favor one task rather than another when they come into conflict, for an arbitrary number of constraints. Similarly, postures are modified in [Baerlocher and Boulic, 2004] by additionally controlling the center of mass (COM). From this work, an efficiently improved technique is applied to motion editing [Le Callennec and Boulic, 2004] (Fig. 2.15, b). Numerical IK algorithms are also applied to motion retargeting approaches which exploit the output of multiple sensors attached to the limbs in order to convert them to a virtual character [Molet et al., 1997]. In [Stolze et al., 1997], the user can additionally control the tradeoff between accurate joint



a **b**

Figure 2.15: (a): The hierarchical motion editing technique applied to a live-captured walking motion . At each stage, the knot spacing grows smaller [Lee and Shin, 1999]. **(b):** Final deformed motion (left), deformed motion without COM constraint (middle) and input motion (right) [Le Callennec and Boulic, 2004].

orientations and accurate joint positions. A more flexible method is proposed in [Monzani et al., 2000] to handle geometrically and topologically different characters, by using an intermediate skeleton. Adjustment is then performed through IK by smoothing joint trajectories before and after the constraint enforcement.

For on-line performance, a hybrid solution based on the inverse rate control technique and joint redundancy is provided by Choi and Ko [2000]. To attain the same objective, Kulpa et al. [2005] propose a motion representation independent of the character's morphology. With this representation, complex constraints can then be rapidly enforced by applying a Cyclic Coordinate Descent algorithm. Finally, other approaches for precise motion synthesis, like manipulation tasks, can combine numerical IK with predefined motion capture postures [Yamane et al., 2004] to ensure realism in the final motion. In [Grochow et al., 2004], a statistical method based on motion capture data allows first the reduction of the motion space dimension, and secondly to define constraints on poses, replacing traditional IK solvers.

While powerful, all these motion editing techniques need improvements to be integrated into an on-line motion generation system, whose footplants are automatically detected and enforced. In this thesis, we introduce a technique to substitute the off-line animator's task, consisting in building up the end-effector trajectories, by an automatic process. These trajectories allow the foot to be re-positioned in order to keep it fixed on the floor level during a constraint.

2.4.3 Motion Anticipation

To deal with end-effector trajectories automatically and in real-time, it is necessary to anticipate the motion in order to obtain future frames of the current animation. Butz et al. [Butz et al., 2003] classify anticipatory mechanisms in four categories: implicit, payoff, sensorial and state-based. We focus on the latter category, dealing with mechanisms in which predictions about future states directly influence current behavioral decision making. In general, anticipation is applied to explore unknown virtual environments, where virtual agents are equipped with multi-sensory systems [Conde and Thalmann, 2004]. A collaborative multi-agent context may also be based on anticipation in order to predict the internal state of other autonomous virtual agents [Veloso et al., 1998]. To our knowledge, only a few works deal with motion anticipation. Labbé et al. [Labbé et al., 2004] propose to anticipate periodic movement trajectories in a prey-predator situation. Steering methods [Reynolds, 1999] can anticipate a motion according to a given desired speed and/or a target to reach. However, these techniques consider the virtual human as a simple 3D model composed of six degree of freedom, instead of a complete articulated system.

To conclude this survey on motion constraints, we summarize the main drawbacks of the previous methods. Firstly, the constraint detection methods are unreliable in an on-line context. Secondly, constraint enforcement methods do not consider the re-positioning of the foot. Finally, anticipation methods based on a real-time locomotion engine do not provide future body postures. In this thesis, we propose a method which aims to provide a complete on-line system to handle motion constraint problems: adaptive constraints detection and smooth constraint re-positioning and enforcement.

2.5 Motion Blending

Motion blending is a technique which combines multiple input motion data according to time-varying weights. It allows to generate either new parameterized motions or transitions by assembling smoothly one animation to another one. The blending techniques give particular attention to the alignment of similar structures of the input motion (time-warping) and to the transition time and duration. In this section, we focus on previous work for motion blending specifically dedicated to the transition generation problems. Note that some techniques discussed in this section have been already presented in previous sections as they combine motion generation with blending.

Precursor works treat the motion as a time-varying signal. Signal processing technique have therefore been developed to perform blending between motions by varying the frequency bands of the signal [Bruderlin and Williams, 1995], or the Fourier coefficients [Unuma et al., 1995]. However, disadvantages appear rapidly: the transition method in [Unuma et al., 1995] is not invertible, support phase of the feet are not considered in [Bruderlin and Williams, 1995]. Other approaches combine motion parameterization with linear blending transitions [Guo and Robergé, 1996; Rose et al., 1998; Golam and Wong, 2000; Park et al., 2002a]. These methods control the blending by changing blend weights of input motions during a period of time given by the user. In addition, similar structures of the blended motions are synchronized in order to avoid blending a right with a left foot hopping action.

Kovar and Gleicher [2003] propose an automatic algorithm for determining these identical motion structures. In addition, the constraints of the blended motions evolve according their assigned weights. However this solution does not allow to add a new input motion dynamically. It needs to be compared to all existing ones in a pre-processing stage. In contrast, Menardais et al. in [Menardais et al., 2004] proposes a synchronization method to perform dynamic blending. If a requested transition is currently unfeasible due to incompatible support phases, the blending is shifted until it becomes possible (Fig. 2.16). Another approach [Ashraf and Wong, 2000] proposes a decoupled blending to resolve diverse coordination configurations between upper and lower body-halves, restricted to cyclic motions. Relevant events of input sequences are manually labeled and correspondences between them are established. Therefore all motions have to be known in advance. In addition, the blending is performed at an interactive frame rate. Another method [Boulic et al., 1997] proposes a weighted joint trajectory blending from basic motions called actions. A set of active joints with a priority is defined for each action, allowing the mixing of a walking with a hand waving action for example.

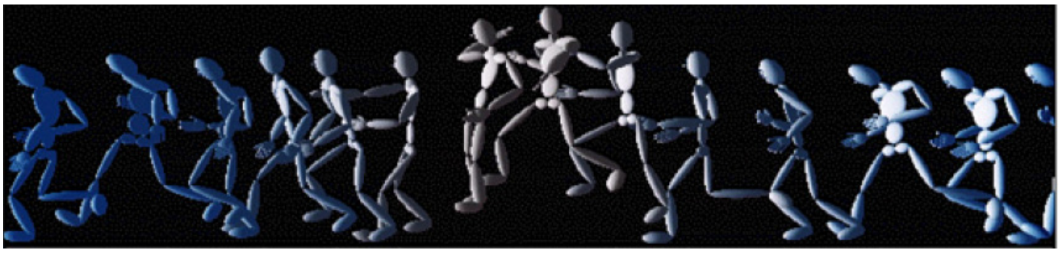


Figure 2.16: Synchronization sequences between running and handball shot [Menardais et al., 2004].

Previous methods offer the possibility to perform transition, but the decision when these can be effectively executed, i.e. the transition time, is let to the animator. This technique is used in the video game industry and is referred to as ”motion graphs” or ”move trees”. An expert user explicitly decided which and when motion clips could be connected. Hence, several works propose to automate this graph construction. Techniques to synthesize seamless streams of video from example footage [Schödl et al., 2000] have been adapted for large motion capture database in order to detect possible transitions between all of these motions. Kovar et al. [Kovar et al., 2002a] analyze posture similarities by comparing pairs of motions, frame by frame. A metric function is defined, based on a Euclidean distance between specific and weighted 3D joint positions. This function measures the distance between two frames: a result smaller than a specified threshold indicates a possible transition between these two frames. A motion graph is then constructed, where edges represent either pieces of original motions or possible (synthesized) transitions, and nodes choice points where motions join seamlessly. The use of this graph allows the animation of a character along a predefined path, with the style of the motion (represented by annotation such as ”lazy walk”) optionally restricted on different parts of the path. A similar approach is presented in [Lee et al., 2002] except that the cost metric distance is based on weighted joint angles and speeds. Then a statistical analysis based on HMM allows the expression of the probability matrices to determine transitions. As an application, the method can generate a motion that best fit a video sequence. In addition, an interface is proposed where a user can directly select what motion is to take place next. Another analogous approach in [Arikan and Forsyth, 2002] builds a hierarchical motion graph and applies a randomized search to extract motions from the graph, which satisfy user-specified constraints such as their durations and joint angles at given key-frames. The metric consists in computing the difference between joint positions and velocities and the difference between the torso velocities and accelerations in the torso coordinate frame. Finally, an extension of this concept of motion graph has been proposed for group animation, called Group Motion Graph [Lai et al., 2005]. Each node of such graphs represents a specific group configuration and edge transition between configurations.

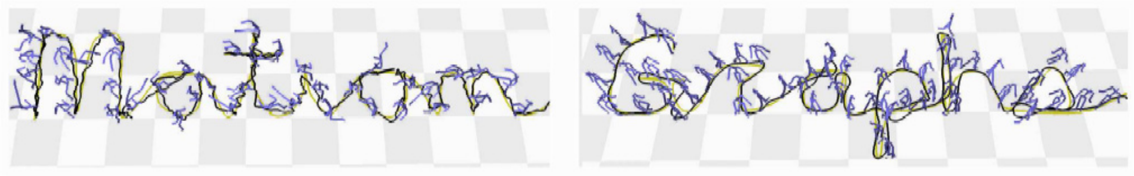


Figure 2.17: A walking motion generated to fit a path using the motion graph technique presented in [Kovar et al., 2002a].

However, motion graphs methods have drawbacks. As pointed by [Arikan and Forsyth, 2002], the definition of a universal metric function is a hard problem. For example, the transition threshold in [Kovar et al., 2002a] has to be manually quantified, as different kinds of motions have different fidelity requirements. In addition, some metric functions contain non intuitive weight values, difficult to parameterize. For example, the proposed set of weights used for the cost metric function in [Lee et al., 2002] is compared to an optimal set in [Wang and Bodenheimer, 2003]. Even if the presented user study - applied on one performer without highly dynamic motions - reveals better results for the optimal set of weights, it remains difficult to evaluate it under various conditions. A second disadvantage concerns the transition duration. Actually, even if the starting point of a transition is found automatically, its duration has to be determined manually. In the study of [Wang and Bodenheimer, 2004], sequences with fixed or variable transition durations have been evaluated by some volunteers. As a result, the users preferred transition durations that are adapted to the context. Thirdly, the on-line insertion of a new motion into the graph is impossible, as all input motions have to be compared pairwise to re-construct the directed graph. This implies that transitions occur only between fixed motions from the database without any possible parameterization. Finally, the motion database must contain only captures from a single character and the produced motion fits this character uniquely. In the next two paragraphs, we discuss methods which tackle those drawbacks.

A number of researchers extend the motion graph concept so as to give more assistance to the users by providing a simpler and more understandable structure. Gleicher et al. [Gleicher et al., 2003] describe how motion graphs can be inappropriate for interactive systems that require fast response times. Since these graphs are unstructured and their connectivity

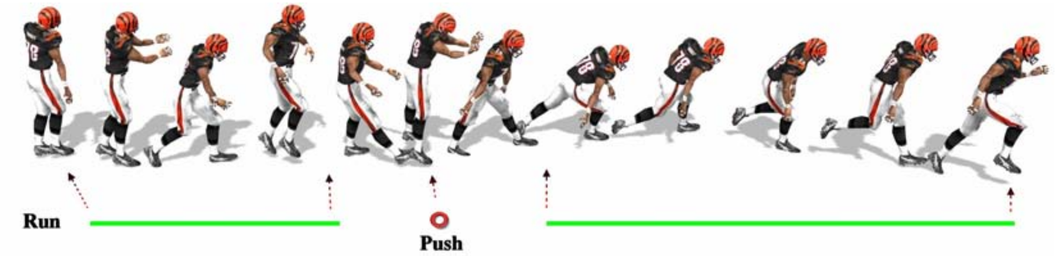


Figure 2.18: The motion is constrained to interpolate the ”push” frame while running before and after the constraint [Arikan et al., 2003].

is complex, it is difficult to know what motions are possible. The authors develop a system which eases the animator tasks by choosing appropriate transitions that can be created around a given posture. In [Arikan et al., 2003] dynamic programming is applied to pieces of motions in order to synthesize off-line a motion from given user annotations. Those can be either geometrical constraint to pass through like forcing a posture to occur at a specific time or combined actions like walking and waving, or running and push at a given frame (Fig. 2.18). For the special case of rhythmic motion synthesis (Fig. 2.19), the method in [Kim et al., 2003] first extracts automatically motion beats and then structures the input motion into a graph, with a classification with respect to their rhythmic parameters. Hsu et al [2004] propose an extended approach to map control specifications (e.g. a *leader* dance motion) to a target motion (e.g. a *follower* dance motion). The method builds a database of control and target segments where those are synchronized motions that together represent a primitive semantic instance of the mapping. Finally, the use of motion graph is evaluated in [Reitsma and Pollard, 2004] for the special problem of character navigation task in an environment. In a first step, the motion graph is pruned to ensure that the generation of motions along a given path does not lead to dead ends. Then the evaluation concerns the ability of resulting motions to follow the shortest path between two points, and the coverage degree of the environment with a given motion graph. As a conclusion, the authors observe that motions of small duration in the database are necessary to reach points in the environment and that a large variety of motion is demanded, like stopping, turning, stepping.

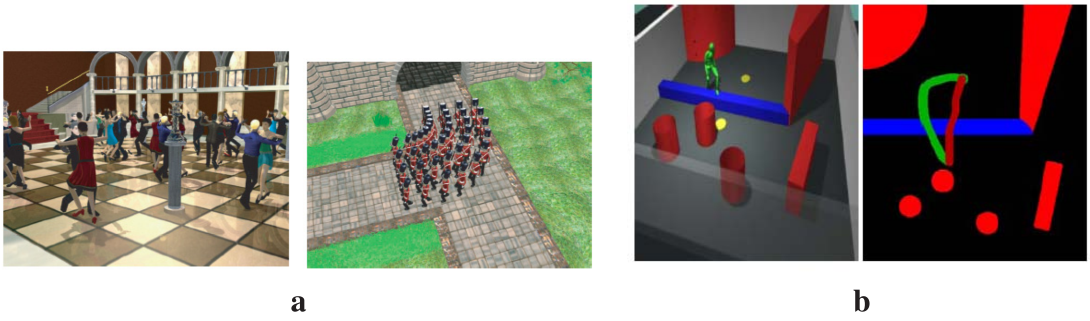


Figure 2.19: **(a):** A ballroom scene and marching soldiers [Kim et al., 2003]. **(b):** A problematic task solved in a unnatural way using motion graph. The red (dark) path is the desired path. The green (pale) path is the shortest path available using this motion graph [Reitsma and Pollard, 2004].

Even if all these blending techniques are widely accepted and used, transitions are not totally controlled and can produce incoherent results. In the study of [Wang and Bodenheimer, 2004], sequences with fixed or variable transition durations have been evaluated by some volunteers. As a result, the users preferred transition durations that are adapted to the context. Approaches taking account for the dynamics like [Rose et al., 1996; Wooten and Hodgins, 2000] can control valid transitions by modifying physical parameters such as angular velocity or center of mass. However, the parameter setting is not straightforward and the motion generation is an off-line process. Recently, a momentum-based method has been developed in [Abe et al., 2004]. Starting from a single motion capture sequence, a space of new physically plausible motions is pre-computed without parameterization effort. Although on-line blending can be performed between these motions, the method is limited to high dynamic movements. Furthermore the transition time and duration are constant. From the earliest work allowing motion parameterization on graph node [Rose et al., 1998], other advanced methods propose to improve the control of transitions. Park et al. [Park et al., 2003] add captured transitions to blend walking, running and standing motions more realistic. Recently, captured transitions for locomotion activities are performed by taking into account an acceleration parameter [Kwon and Shin, 2005]. However, the available transitions inside a node are not sufficient enough to avoid abrupt blending.

To conclude this overview dedicated to motion constraints, three main drawbacks emerge. Firstly, the constraint detection methods are unreliable in an on-line context. Secondly, constraint enforcement methods do not consider the re-positioning of the foot. Finally, anticipation methods based on a real-time locomotion engine do not provide future body postures. In this thesis, we propose a method which aims to provide a complete on-line locomotion system to handle motion constraint problems: adaptive constraints detection and smooth constraint re-positioning and enforcement.

2.6 Motion Planning

Motion planning increases the autonomy of virtual character. Generally, given initial conditions and a goal, a path planner method provides a path to follow. Concerning the special case of locomotion tasks, the planner provides the path taking into account the obstacle avoidance problem. Additionally, this path is transformed into a trajectory which gives the adequate locomotion parameters at each time step. We focus on this specific problem by listing appropriate approaches.

The probabilistic roadmap methods (PRM) sample the configuration space randomly at preprocessing time, to construct an accessible point graph called a roadmap. This latter is then used to search for a path during the planning stage [Kavraki et al., 1996]. These methods have empirically demonstrated good performances for difficult problems. Moreover, this approach is generic, applicable to navigating car-like robots with non-holonomic constraints, humanoid robots with many degrees of freedom, but also molecules movements in biochemistry.

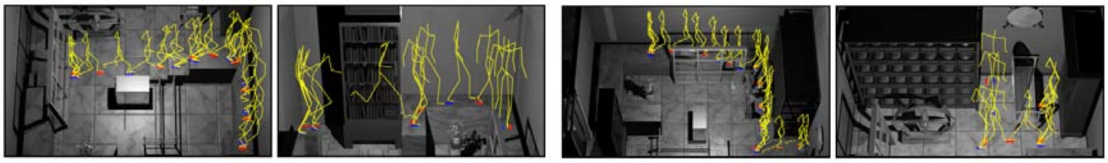


Figure 2.20: Resulting motion constructed with the footprints (red and blue) computed by the path planning [Choi et al., 2003].

Pettré et al. [Pettré et al., 2003a,b] apply PRM to get a collision-free 2D path in a 3D environment. This path represented by Bezier curves generates a trajectory encapsulating linear and angular speed variations. Finally a walking animation is generated according to those variations. Another approach based on PRM is presented in [Choi et al., 2003]. The roadmap is constructed by randomly sampling foot positions and orientations to form the graph nodes. The connection between two nodes is possible if a subsequence searched in a motion capture database approximates the given foot configuration well. The final animation from a given start and goal position is obtained by traversing the graph. The input motion clips are adapted to the foot configuration described in the graph nodes (Fig. 2.20). Recently, a similar method [Sung et al., 2005] has been proposed for crowd simulation. It uses a motion graph structure to obtain approximate collision-free motions which match constraints requiring a characters to be in specific poses, positions and orientations at specified times. Those motions are then refined through a fast randomized search algorithm. However, these techniques are only designed for static environment and are not adapted to real-time context. Other footprints approaches can be applied to generate animation given a PRM. Tsumura et al. [Tsumura et al., 2001] present a locomotion method adapted to abrupt direction changes by using the next footprints positions. However, this method does not control the locomotion speed, and results in coarse motion quality. Van de Panne [1997] proposes to compute the leg motion of an articulated figure given a set of footprints with their timing. However, this approach is not real-time and not tested on human-like figure.

Other approaches based on A* algorithm allow to search a path into a tree structure at interactive or even real-time performance. Kuffner in [Kuffner, 1998] proposes to subdivide the 2D scene projection into a regular grid of cells. Each cell is labeled as free or occupied by an object. A collision-free path is then search using a dynamic programming algorithm such as A*. The final animation is generated using only a single walking cycle played at varying frequency according to a proportional derivative controller. However, due to the 2D projection, the obstacles are considered as a wall and can therefore not be stepped over.

Recently, finite state machine (FSM) have been used to abstract different behaviors [Lau and Kuffner, 2005; Chestnutt et al., 2005]. From a current behavior state, the FSM proposes a set of possible next behavior states to be reached. The optimal one is then chosen by applying an A* search algorithm so as to find the solution which is less costly. Hence, the results are highly dependent on the database size. In addition, the approach in [Lau and Kuffner, 2005] allows to jump over obstacles (Fig. 2.21). However, those have to be assigned to a specific motion clip. It is therefore not possible to add an obstacle dynamically in the scene without having its corresponding jumping motion in the database.

The previous approaches are not well adapted for dynamic scene where the number and

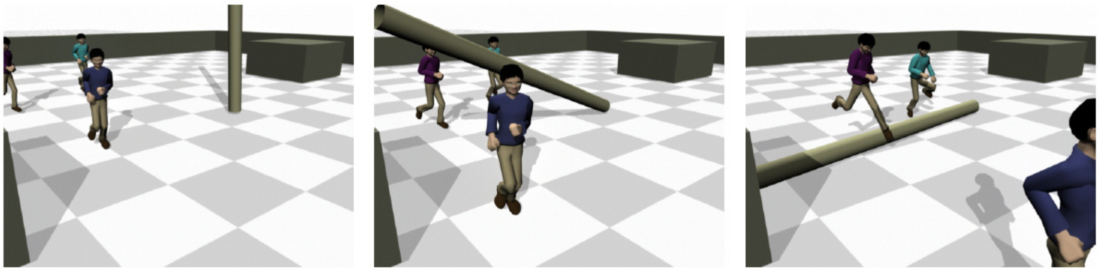


Figure 2.21: A dynamic environment with a falling tree. **Left:** Before it falls, the characters are free to jog normally. **Center:** As it is falling, the characters can neither jog past nor jump over it. **Right:** After it has fallen, the characters can jump over it [Lau and Kuffner, 2005].

position of the obstacles are not known in advance. In [Park et al., 2002b], the method proposes to modify the computed path jammed by a moving obstacle. The solution is based on the work of Badler et al. [Badler et al., 1996] using repulsive force assigned to the obstacle. However, this technique is off-line and was applied only to the hand movements. Kathib et al. [Brock and Khatib, 2002] introduce a method which represents the path as a deformable elastic strip. From a previously planned motion, the path can be modified in real-time according to moving obstacles intruding it. Originally designed for robot motions, this method can be applied to virtual human. However, the skeleton is controlled dynamically, reducing the complexity of motion activities (e.g. ski).

Finally, the approach in [Go et al., 2004] allows the steering of a 3D point into a dynamic environment by selecting the best trajectory among predefined ones. However, this technique is well adapted for 3D point animation such as a spacecraft, but still difficult to adapt for virtual human whose motion constraints are much more complex. In fact, the pre-computed trajectories have a constant linear speed.

In this thesis, we improve the previous work by addressing the problem of dynamic obstacle avoidance by the generation of appropriate motions on-the-fly, instead of searching into a database of pre-computed motion sequences. As the obstacle dimension and position are unknown in advance, the trajectory has to be computed not only efficiently to preserve real-time capability of the animation system, but also precisely in order to prevent any collision between the character and obstacle.

Part II

Motion Modeling

Chapter 1

Fundamentals

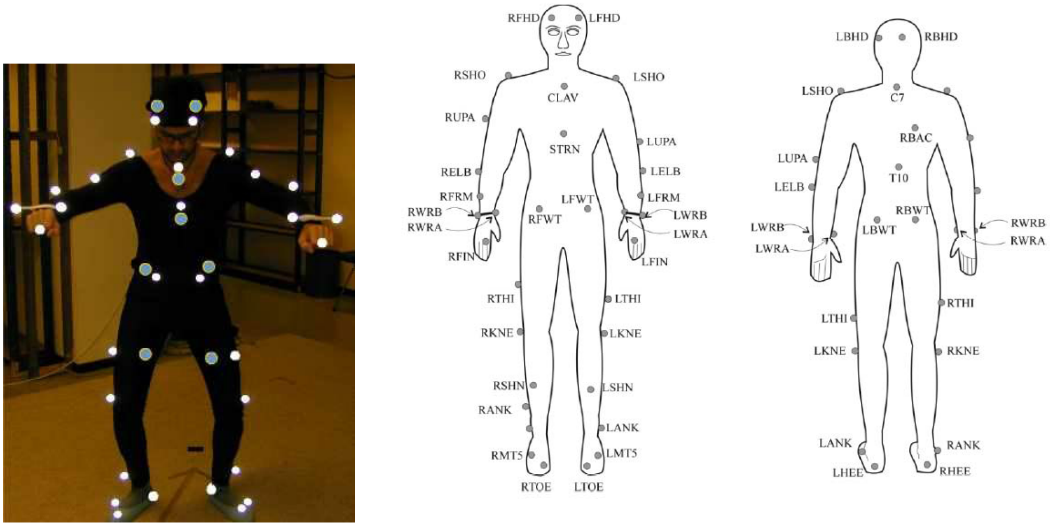
The modeling of human motion consists in describing a formal system based on laws derived from real observations. This system, driven by parameters, aims at the synthesis of new motions. The human motions are so diverse that it would be unrealistic to imagine a unique model. In this thesis, the motion modeling is therefore restricted to two basic human activities: the locomotion, from walk to run, and the jump activity. First, we focus on the essential material necessary to provide such a model (or system). In this chapter, the objectives consist in describing the acquisition of real motions, and choosing a method facilitating the elaboration of the system laws.

1.1 Data Acquisition

Various technologies allow the acquisition of real motion data. In the field of Biomechanics, instruments such as goniometers, accelerometers or the electromyography (EMG) are used to get specific motion information. However, it is still difficult to describe precisely the corresponding human motion from that data. We opt for another technology, namely the motion capture. This choice is motivated by its main advantage: the high level of realism. Our setup is composed of an optical motion capture system [Vicon Motion System®, 2005] with eight cameras working in the infrared spectrum at a frame rate of 120 Hz. The recorded performances are executed by an actor who wears a stretch suit with reflective markers, as illustrated in Fig. 1.1 (a). They are placed at strategic locations, in order to reconstruct best quality joint movements.

The motion acquisition protocol proceeds as follows. For locomotion, this activity is performed on a treadmill. In order to be more familiar and comfortable with it, the actors (or subjects) were first asked to train during approximately 5 minutes before being captured. Then walking and running sequences of approximately one minute have been recorded, by setting a constant treadmill speed for each take (Fig. 1.2). In the end, the locomotion database is composed of motions which differ from three parameters: the subject, type of locomotion and speed.

The validity of performing treadmill instead of overground locomotion has been an issue



a b

Figure 1.1: (a): An actor wearing the stretch suit. **(b):** From the Vicon user's guide: the location of reflective markers (front and back side).

of debate for many years. In short, the treadmill walking induces a shorter stride length and increased cadence [Stolze et al., 1997] as well as running [Schache et al., 2001]. The reasons are many: anxiety, caution, lack of optical flow. The material (e.g. the walking surface) and mechanical (e.g. belt speed fluctuations) properties of the treadmill is also associated to kinematic and dynamic variations in motions. Moreover, motorized treadmills can significantly affect the variability and local stability of gait. Dingwell et al. [Dingwell et al., 2001] demonstrate that the locomotor variability is reduced while improving the stability.

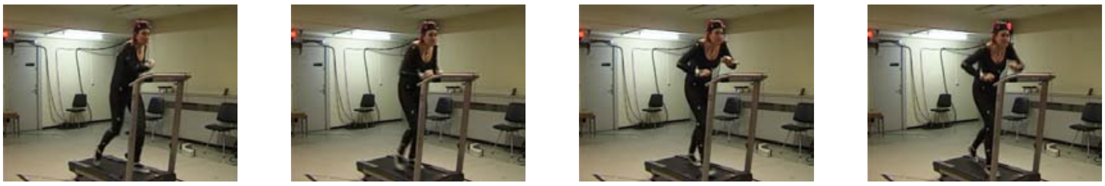


Figure 1.2: Motion capture session of walking on a treadmill.

For jump activities, the acquisition protocol is the following. We asked the subjects to perform long jump, either with a walking or a running run-up, in order to clear an obstacle. This latter is represented by two small boxes, one indicating the take-off foot position and the other the landing foot position (Fig. 1.3). For our experiment, we limit the capture by recording only jumps with right take-off foot. The subjects had to adapt their run-up speeds so as to execute the requested jump length as most naturally.

As the motion capture data technique produces noisy data, a cleanup stage is necessary, executed using a commercial software [Vicon Motion System®, 2005]. The motion sequences have to be free from artifacts in order to validate the applied motion model correctly. Actually, the use of bad quality sequences corrupts the final results. The validation of

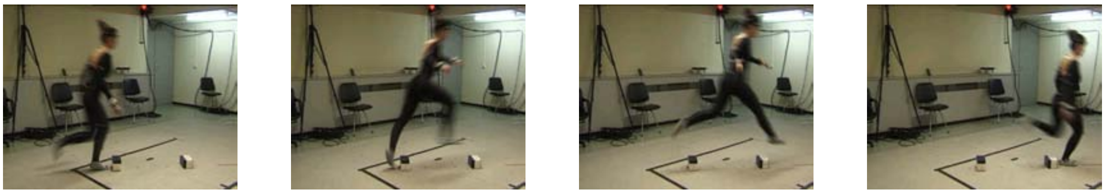


Figure 1.3: Motion capture session of jumping.

those results becomes difficult if we have to distinguish drawbacks between the input motion and the motion model. The cleanup stage consists first in ensuring the marker trajectory continuity. A marker trajectory can be interrupted during a time interval on account of occlusion. The motion capture system may also swap the trajectories of two near markers. The second point concerns the elimination of marker trajectories called ”phantom” which come from false reflection due to the presence of reflective material in the capture area. Finally, all marker trajectories are filtered to produce smooth curves.

Additionally to this cleanup stage, the motion sequences are manually split so as to obtain a ”**motion unit**”. In case of locomotion, we define the motion unit (or cycle) as the sequence of two steps, beginning and ending with the same foot event. We arbitrarily choose this event as the right heel strike. As these motion units are cyclic, the beginning posture has to match the ending one. This process, referred to as ”cyclification” is performed using a commercial software [MotionBuilder®, 2005]. More sophisticated cyclification approaches (fine parameterization, automation) can be implemented [Ahmed et al., 2003; Wagner da Silva et al., 1999]. However, in our situation, the commercial solution is enough satisfying. The jump motion unit is also determined by the sequence from the right heel strike event to the next same event. Hence, this sequence holds the three phases of a jump: the take-off, flying and landing. Note that all motion units are expressed as marker 3D positions varying over time.

1.2 Principal Component Analysis

The motion units described in the previous section constitute the motion database which forms the real data observations used in our motion model. In order to define this model, we propose a method facilitating the elaboration of its laws. The method has to be capable of working with our database, composed of a large number of motions, each being represented by a succession of frames (or postures).

With this aim in mind, the choice of motion representation is important. In [Rose et al., 1998], the joints are represented by Euler angles. The authors approximate the joint animation curves with a uniform cubic B-spline specified by a determined number of control points. Other techniques [Kovar and Gleicher, 2003; Park et al., 2002a] use directly the joint orientations at each frames, expressed with quaternions. We choose this representation approach by exploiting the Principal Component Analysis (PCA). As this algorithm reduces drastically the data, we can abstract the input motion as a single entity, as opposed to previous methods dividing the animation frame by frame [Alexa and Müller, 2000; Alexa, 2002].

PCA is a wide-spread statistical method whose central idea is to reduce the dimensionality of an input dataset which contains certain relation between its different variables. Even though one of the first explanations of the PCA has been published early in the 20th century, it is only since the 80s that this technique becomes popular [Jolliffe, 1986]. Many areas take advantage of PCA: biology, chemistry, climatology, demography, ecology, economics or psychology. Since the mid 90's, Computer Vision and Computer Graphics have also demonstrated a growing interest in this subspace analysis technique, particularly in shape parameterization and animation.

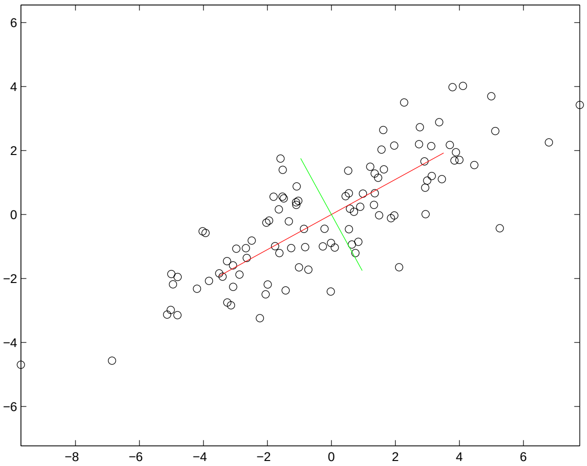


Figure 1.4: Result of a PCA applied on a 2D dataset. In red, the first principal component, in green the second (orthogonal to the first).

Essentially, PCA transforms a set of correlated variables into a set of uncorrelated variables which are ordered by reducing variability. The uncorrelated variables are linear combinations of the original variables. As PCA reduces the dimension of the original data, the coefficient number of the linear combination can be smaller than the number of variables with minimum loss of data. Hence it produces a more compact and better description of the data, as shown on Fig. 1.4. The first Principal Component (PC) explains the greatest amount of data variation; the second principal component defines the next largest amount of variation and is orthogonal to the first principal component, and so on until having reached a sufficient number of principal components to describe the original dataset.

The Appendix B introduces some basic statistical and linear algebra background, as well as the PCA technique necessary to understand the next paragraph focused on PCA applied to motion data.

1.3 A PCA Algorithm for Motion Data

We present the PCA algorithm applied on our capture motion data, where a motion is considered as a sample. We start with an example composed of n samples being time normalized to get an identical fixed frame number N_{frame}. A sample, composed of frames $\mathcal{F}_i$, is defined as a motion vector $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} = (\mathcal{F}_1, \ldots, \mathcal{F}_{N_{frame}}) \tag{1.1}$$

where its dimensions are $(1 \times p), p = N_{frame} \cdot d$, where d represents the frame dimension. The p value $(p > n)$ corresponds to the number of observation variables whose variance have to be minimized by the PCA. The PCA algorithm runs as follows:

1. Construct a motion matrix $\mathbf{M}$ of dimension $(p \times n)$ so that the i-th column contains a motion vector $\boldsymbol{\theta}_i$.

2. Compute $\boldsymbol{\theta}_0$ as the average vector of all n motion vectors, in order to center the resulting reduced space with respect to the whole dataset. A new motion matrix $\widetilde{\mathbf{M}}$ is therefore defined as:

$$\widetilde{\mathbf{M}} = (\boldsymbol{\theta}_1 - \boldsymbol{\theta}_0, \boldsymbol{\theta}_2 - \boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_n - \boldsymbol{\theta}_0) \tag{1.2}$$

3. Compute the eigenvectors $\mathbf{e}_i$ and the eigenvalues λ_i, $(i = 1 \ldots p)$ from the covariance matrix $\mathbf{C} = \widetilde{\mathbf{M}}\widetilde{\mathbf{M}}^T$. Note that for $i = [n+1, n+2, \ldots, p]$, $\lambda_i = 0$.

4. Approximate a motion vector $\boldsymbol{\theta}$ with the eigenvectors associated to the first k largest eigenvalues as follows:

$$\boldsymbol{\theta} = \boldsymbol{\theta}_0 + \sum_{j=1}^{p} \alpha_i \mathbf{e}_i \approx \boldsymbol{\theta}_0 + \sum_{j=1}^{k} \alpha_i \mathbf{e}_i = \boldsymbol{\theta}_0 + \boldsymbol{\alpha}\mathbf{E}^T \tag{1.3}$$

   where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_k)$ is a coefficient vector and $\mathbf{E} = (\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_k)$ a matrix of the first k PCs (or eigenvectors).

In practice, the eigenvalue computation of the covariance matrix can be generalized to the problem of eigenvalue decomposition of a matrix $\mathbf{C} = \mathbf{A}\mathbf{A}^T$. Substituting the singular value decomposition (SVD) on the matrix $\mathbf{A} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^T$ in $\mathbf{A}\mathbf{A}^T$, we can observe that the eigenvalues of $\mathbf{A}\mathbf{A}^T$ (same as these of $\mathbf{A}^T\mathbf{A}$) can be computed using the singular values held in $\boldsymbol{\Sigma}$. Furthemore, the eigenvectors of $\mathbf{A}\mathbf{A}^T$ are contained in $\mathbf{V}$ (and $\mathbf{U}$ for $\mathbf{A}^T\mathbf{A}$). Hence, the eigenvalues and eigenvectors computation can be computed in two ways. The first one consists in applying SVD on the matrix $\mathbf{A}$ of type $(n \times p)$. To improve the computational speed-up, SVD on $\mathbf{A}^T$ is preferred if the are less columns than lines (i.e. $n < p$) [Press et al., 1992], as described in Algo. 1. The other alternative is based on the eigenvalue decomposition. This process is optimized by selecting between $\mathbf{A}\mathbf{A}^T$ and $\mathbf{A}^T\mathbf{A}$ the matrix having the lowest dimension. Actually, the matrix $\mathbf{A}^T\mathbf{A}$ has the same eigenvalues λ_i as $\mathbf{A}\mathbf{A}^T$ (the remaining $n - p$ values equal zero), and its eigenvectors $\mathbf{u}_i$ are an expression of the eigenvectors $\mathbf{v}_i$ of $\mathbf{A}\mathbf{A}^T$ (see Appendix B):

$$\mathbf{v}_i = \frac{1}{\sqrt{\lambda_i}} \mathbf{A}\mathbf{u}_i \tag{1.4}$$

The complete process is summarized in Algo. 2.

Algorithm 1 Singular Value Decomposition

```
if $nb_{col} \geq nb_{line}$ then
    svd(A, V, S, U)
    EigVa = diag(S)*diag(S)
    EigVe = V
else
    svd($A^T$, V, S, U)
    EigVa = diag(S)*diag(S)
    EigVe = U
end if
```

Algorithm 2 Eigenvalue Decomposition

```
if dim($AA^T$) $\leqslant$ dim($A^TA$) then
    EigVa = eigenvalues($AA^T$)
    EigVe = eigenvectors($AA^T$)
else
    EigVa = eigenvalues($A^TA$)
    EigVe = (1/sqrt(EigVa))*A*eigenvector($A^TA$)
end if
```

Instead of using the covariance matrix, its ”normalized” form, namely the correlation matrix, can be chosen. However, the eigenvalues and eigenvectors of the correlation matrix do not have a simple relationship with those of the corresponding covariance matrix. The PCs for correlation and covariance matrices do not give therefore equivalent information.

A major argument for using correlation matrix is that PCA based on covariance matrices produces PCs sensitive to the unit of measurement used for each variable of the original data. If the observed variables are expressed in different measurement units, it implies a large difference between their variance, and the variables whose variances are largest would tend to dominate the first few PCs. The resulting PCA space is therefore conditioned by the magnitude of the variables instead of their intrinsic relations.

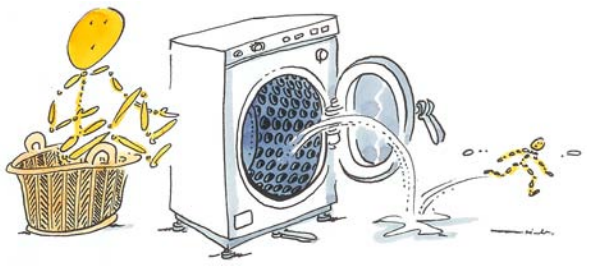
However, statistical inference is difficult on PCs computed with correlation matrices. In fact, the principal coefficients are given for standardized variables and are therefore less easy to interpret directly. In practice, covariance matrices are preferred if the measurement unit are identical for all sample variables. Though as a preventive measure, the standard deviations of all variables are compared. Large differences between them alert that considerable differences exist between the PCs for the correlation and covariance matrices. In that case, the contribution c_i of each i-th PCs for both alternatives is measured as follows:

$$c_i = \frac{\lambda_i}{\sum_{j=0}^{p} \lambda_j} \tag{1.5}$$

If the contributions of the first few PCs computed with the covariance matrix are much significant than those from the correlation matrix, the covariance option is preferred.

1.4 Conclusion

In this chapter, we have described the acquisition process of real motions necessary for the motion modeling. An optical motion capture system is used to capture walking, running and jumping sequences, varying by parameters such as for example the performer, the speed, the jump run-up and length. The motion data has been carefully post-processed so as to be free from noise and have been segmented in motion units. We have also described the PCA method which is able to reduce the dimension of the motion database by the generation of a new space. The first dimension of this resulting PCA space explains the greatest amount of data variation, the second dimension the second greatest variation, and so on. This reduced space is therefore very appropriate to identify laws which have to describe the observations, i.e. our variety of motions. We have also analyzed the motivations to choose between the covariance and correlation matrix for the PCA computation.



Chapter 2

Generation of Motion Units

2.1 Introduction

This chapter is dedicated to the identification of laws characterizing the motion, by applying PCA to the captured (original) motion units described in the previous chapter. Those laws, driven by parameters, aim at the generation of new motion units, either inside the convex hull spanned by the original motions (motion interpolation) or beyond it (motion extrapolation). The law parameters have to describe the motion at a high-level in order to provide adequate assistance to animators using this model. In addition, the parameter dimension is supposed to be large enough to produce diversified motion units.

The elaboration of these laws is based on the following methodology:

- First we choose a motion representation for our data which we want to observe. The choices are motivated by the nature of the motion capture data and the potential of a motion representation for PCA application.
- PCA is then applied on this motion data, by considering a motion as a single observation. We decide either to compute the covariance or the correlation matrix, depending on the input data. In addition, the contribution (or influence) of the PCs is evaluated to determine the dimension of the resulting PCA space.
- This reduced space is analyzed in order to draw laws which characterize our motion model. The strategy is to relate the coefficients (or scores) of a motion, expressed in this space, to its parameters. Those relationships are visualized on 2D graphs to help the analysis.

In the next sections, we describe step by step this methodology, applied to our motion capture data.

2.2 PCA on Angular Data

The very straightforward idea when dealing with motion capture data is to use directly the 3D marker trajectories. However, this approach does not ensure that the distance between the connected markers remains constant, as the PCA approximates the 3D marker trajectories. By experiments, the modification of an original motion by the variation of its Principal Coefficient values emphasizes this approximation. Hence, the resulting motion may violate basic properties of the human movements: joint limits, floor penetration and gravity force. To tackle this problem, some controllers can be added to correct the generated motions. However, this solution is difficult to implement, as the correction to apply is very significant.

For those reasons, we choose another motion representation. We convert our motion data into the joint angle space. The 3D marker trajectories are mapped on a human skeleton 2.1. This latter is represented using the standard H-Anim [H-ANIM, 2005] which describes a hierarchical skeleton with specific joint names. The use of the joint angle space is motivated by two important points. First, the length of the limbs remains constant, independently on the joint angle values. Secondly, it is possible, for a generated joint angle value, to check if it is out of some joint limits [Korein, 1985]. Therefore, this motion space is more appropriate to handle the problems encountered with the motion representation based on 3D marker positions.

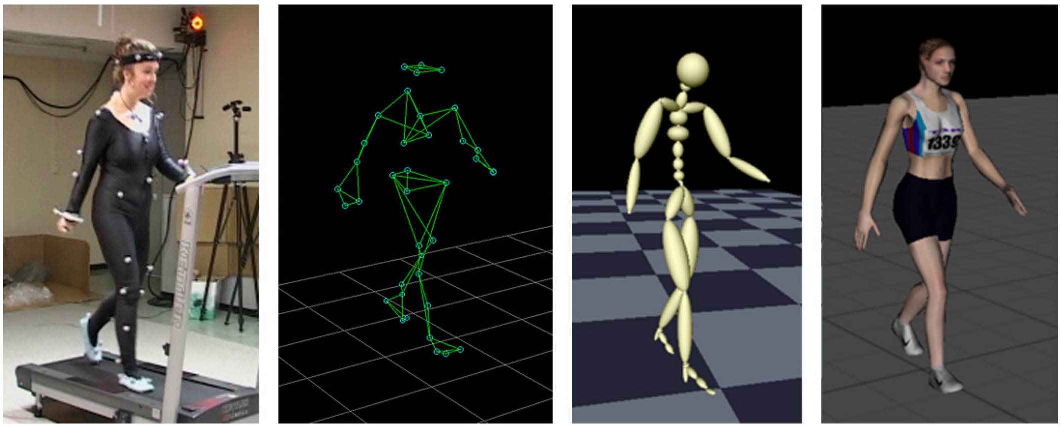


Figure 2.1: A posture from real to virtual human. **From left to right:** Real performer; 3D positions recorded by motion capture; converted positions into the joint angle space map onto a simple character; and onto a skinned virtual human.

We have to choose over the various joint parameterizations which have been put forward by the graphics community over the last two decades. Their suitability with respect to different applications are discussed by Grassia in [1998]. He concludes that there is no ideal parameterization and emphasizes that no single parameterization is free of singularity in R^3. By contrast, unit quaternions and rotation matrices avoid singularities (i.e., they are safe from gimbal lock) since they work in a different space. We opt for the compact quaternion parameterization. A rotation is represented by a unit quaternion which lies on the hypersphere S^3 embedded in the four-dimensional Euclidean space R^4.

The application of PCA on joint rotations is problematic. PCA is a linear technique and therefore well adapted for linear data. However this technique reaches its limits when the data is nonlinear, for example the joint rotations which are commonly represented with quaternions in a non-Euclidian space. Therefore, our original motion data which is represented by quaternions, has to be converted into a parameterization embedded in R^3. This parameterization is referred to as exponential maps and is introduced in [Grassia, 1998]. This technique, based on a tangent space, maps a vector in R^3 describing the rotation axis $\mathbf{v}$ and the magnitude ϕ (i.e. $||\mathbf{v}||$) of a three DOF rotation. Hence, a unit quaternion $\mathbf{q}$ can be formulated as follows:

$$\mathbf{q} = \exp(\mathbf{v}) = \left[\sin\left(\frac{1}{2}\phi\right)\frac{\mathbf{v}}{\phi}, \cos\left(\frac{1}{2}\phi\right)\right]^T \tag{2.1}$$

Using Eq. 2.1, $\mathbf{q}$ can be mapped into the log space:

$$\ln(\mathbf{q}) = \mathbf{v} \tag{2.2}$$

We represent our motion data with exponential maps in order to bring all the benefits of a Euclidean parameterization: addition is really addition for example. However, this parameterization is subject to the following caveat [Grassia, 1998]. Either the considered exponential maps must have ”small” norms, or their axes of rotation must not diverge ”too much” from parallel.

We illustrate this limitation by defining 25 orientations $\mathbf{q}_{yi}$ around the Y-axis (from $-\pi$ to π) and 25 similar orientations $\mathbf{q}_{zj}$ around the Z-axis (from $-\pi$ to π). For each pair of $\mathbf{q}_{yi}$ and $\mathbf{q}_{zj}$, the mean orientation is computed in two ways: first using the SLERP interpolation ([Shoemake, 1985]) between two quaternions (m_{slerp}), and secondly using the linear interpolation with the two corresponding exponential maps $\mathbf{v}_{yi}$ and $\mathbf{v}_{zj}$ (m_{expmap}), as described in Eq. 2.3:

$$\begin{aligned} m_{slerp} &= \ln\left(\text{SLERP}\left(\mathbf{q}_{yi}, \mathbf{q}_{zj}, 0.5\right)\right) \\ m_{expmap} &= \frac{||\ln(\mathbf{q}_{yi}) + \ln(\mathbf{q}_{zj})||}{2} \end{aligned} \tag{2.3}$$

Fig. 2.2 shows the difference between those two means, for each pair $\mathbf{q}_{yi}$ and $\mathbf{q}_{zj}$. This difference increases as the orientations are far from zero, indicating that the linear interpolation becomes less appropriated.

The frames $\mathcal{F}_i$ ($1 \leq i \leq N_{frame}$) which compose a motion vector $\boldsymbol{\theta}$, as described in Eq. 1.1, are therefore defined as follows:

$$\mathcal{F}_i = (\widehat{\mathbf{p}}_{ri}, \ln(\widehat{\mathbf{q}}_{ri}), \ln(\mathbf{q}_{1i}), \ldots, \ln(\mathbf{q}_{mi})) \tag{2.4}$$

where $\widehat{\mathbf{p}}_{ri}$ is the translation and $\widehat{\mathbf{q}}_{ri}$ the quaternion of the character's root, and $\mathbf{q}_{ji}$ for $j > 0$ the local transformation of the j-th body's joint. In our configuration, we use a skeleton composed of 26 joints, leading to a body having 40 DOFs.

To validate the use of exponential maps for our motion data, we compute their norms in all $\boldsymbol{\theta}$ from our database, for each joint and at each frame $\mathcal{F}_i$. From all observed joints, we note that exponential maps related to the knee joint contain the most important norm variation

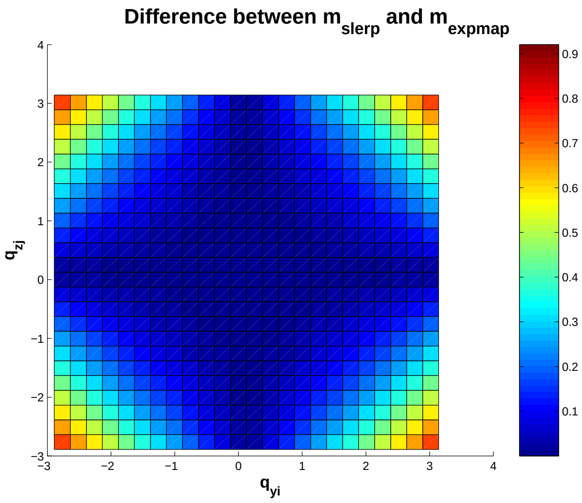


Figure 2.2: The difference between m_{slerp} and m_{expmap} for each pair $\mathbf{q}_{yi}$ and $\mathbf{q}_{zj}$.

(see Fig. 2.3, left). We have therefore to make sure that their corresponding axes of rotation are "near" from each others. At each frame $\mathcal{F}_i$, the maximal angle between those axes of rotation is computed (see Fig. 2.3, right). It results that for a significant norm of exponential maps at a given frame, the corresponding axis are "near" from each others. Hence, we are allowed to apply Euclidean operations to our motion data represented by exponential maps.

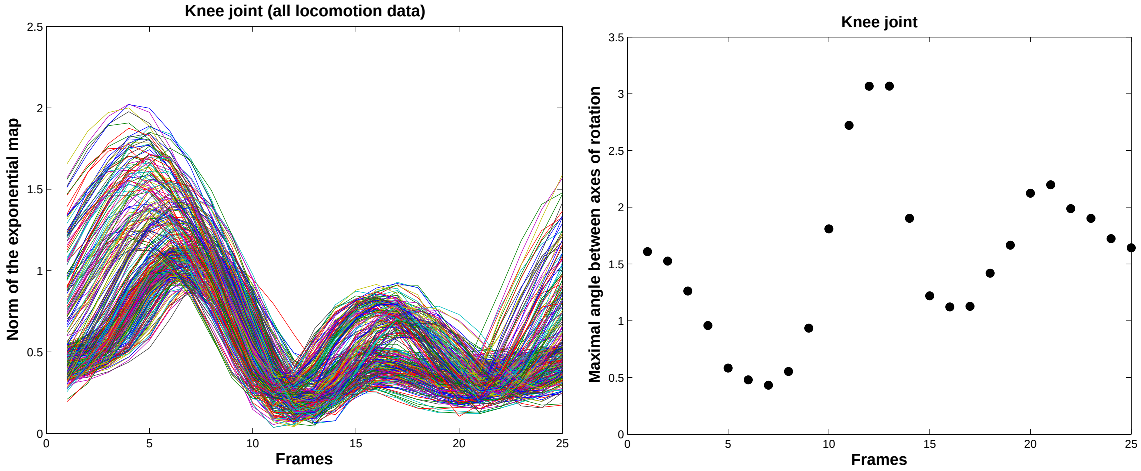


Figure 2.3: Left: Each curve represents the evolution of the norm of the exponential map related to left knee joint, for a given motion. **Right:** The maximal angle between all axes of rotation of exponential maps representing the knee joint (for all $\boldsymbol{\theta}$ of our locomotion database) at each frame $\mathcal{F}_i$.

2.3 Walking and Running Motion Units

In this section, we focus on the two next points of our methodology described in the introduction of this chapter: the application of the PCA and its analysis to draw laws which characterize our motions. We first apply PCA either using covariance or correlation matrix. Then we select the number of PCs and finally we analyze the resulting PCA space to establish relationship between coefficients and motion parameters.

We use the locomotion capture database described in the previous chapter (Section 1.1). In the next two sub-sections, we present our motion model. For a better understanding of our approach, we propose a simplified situation in the following sub-section (2.3.1). In fact, we focus on walking sequences captured on individual performers in order to observe one high-level motion parameter: the speed. Then, in the second sub-section (2.3.2), we introduce more variety by using our complete locomotion database. Hence, two high-level parameters are added: the type of locomotion (either walk or run) and personification (the styles of the captured subjects).

2.3.1 Single High-level Parameter Variation

The underlying idea is to identify laws able to generate a parameterized motion vector $\boldsymbol{\theta}(s)$, or more precisely a walking cycle, where s represents a locomotion speed value. This aim in mind, a PCA space is computed by applying the PCA algorithm described in the previous chapter (Section 1.3) on a matrix $\widetilde{\mathbf{M}}^k$, computed by subtracting the mean motion $\boldsymbol{\theta}_0$ to each column of $\mathbf{M}^k$. This matrix is composed of $N_{seq}^{walk,k}$ samples. They correspond to various walking motion vectors $\boldsymbol{\theta}$ from a specific subject k, and to a standing posture. This posture is necessary in order to obtain a motion sample corresponding to the null speed. The p sample variables whose variances have to be maximized are the three components of the global translation $\widehat{\mathbf{p}}_{ri}$ and orientation $\ln(\widehat{\mathbf{q}}_{ri})$, as well as the other orientations $\ln(\mathbf{q}_{ji})$ attached to the body, for a given i-th frame.

PCA is either performed with covariance or correlation matrix. We choose the covariance matrix as the measurement unit is the same for all elements of the data, except the global motion $\widehat{\mathbf{p}}_r$. Nevertheless, a further comparison between the standard deviation of the p sample variables is performed. The computation of the standard deviation σ_{p_i} for a given variable p_i over j observations is executed as follows:

$$\sigma_{p_i} = \sqrt{\frac{1}{n}\sum_{j=0}^{n}(p_{i_j} - \bar{p_i})^2} \tag{2.5}$$

where $\bar{p_i}$ is the mean of all p_{i_j} observed. An observed p_i sample variable represents a component of either $\widehat{\mathbf{p}}_r$, $\ln(\widehat{\mathbf{q}}_r)$ or $\ln(\mathbf{q})$.

We observe on Fig. 2.4 that the standard deviation relative differences are not considerable. The larger differences correspond to specific joint whose rotations change significantly between the samples: the knees, the ankles and the elbows. Furthermore, the single variable measured in another unit, namely the global motion $\widehat{\mathbf{p}}_r$, has a similar variance magnitude as the other joints. The same observations are also consistent for the other subjects. The covariance matrix is therefore a correct choice.

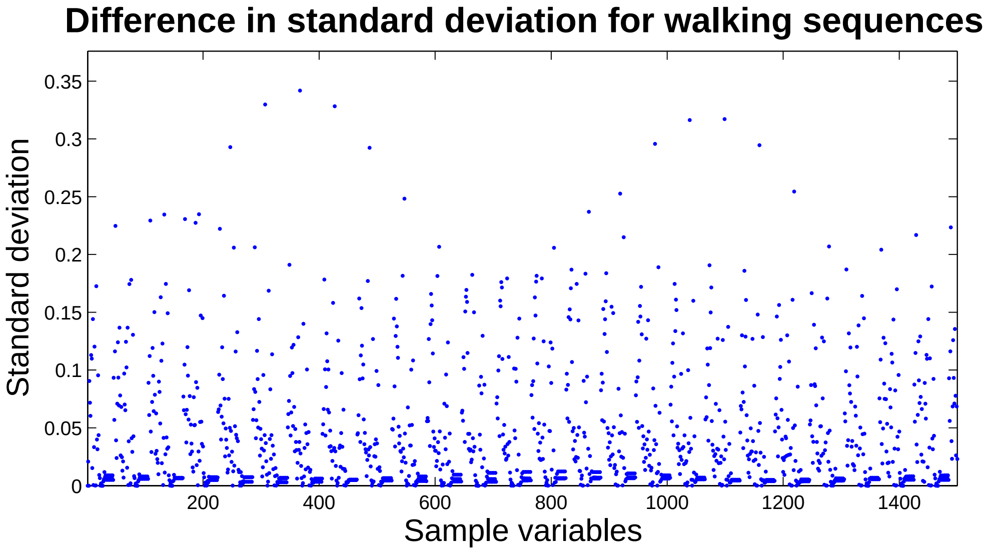


Figure 2.4: . The standard deviation of each variable of the walking motion matrix $\widetilde{\mathbf{M}}^k$, for a given subject k.

The PCA algorithm applied to the covariance matrix of $\widetilde{\mathbf{M}}^k$ returns a matrix $\mathbf{E}^k$ containing the first m PCs (with $m \leq N_{seq}^{walk,k}$) that are orthogonal vectors (or eigenvectors) describing the PCA space, and a vector $\boldsymbol{\alpha}^k$ holding the coefficient values of these first m PCs. In this new space, a motion $\boldsymbol{\theta}$ can be approximated by:

$$\boldsymbol{\theta} \approx \boldsymbol{\theta}_0 + \sum_{i=1}^{m} \alpha_i^k \mathbf{e}_i^k = \boldsymbol{\theta}_0 + \boldsymbol{\alpha}^k \mathbf{E}^{k^T} \tag{2.6}$$

The number m of the first PCs, determining the dimension of the PCA space, has to be fixed. As illustrated in Fig. 2.5, the contribution of the first PCs to the total motion information is very important, while other PCs do not provide relevant relation. Therefore, we choose to set m so that PCs represent more than 80% of the original information.

As only the speed parameter differs in the motion matrix $\widetilde{\mathbf{M}}^k$, the resulting PCs $\mathbf{e}_i^k$ tend to characterize the variance between slow and fast motions. Therefore, our goal is to find a relationship $\boldsymbol{A}^{walk,k}(s)$ that maps the coefficient vectors $\boldsymbol{\alpha}^k$ describing our original motions with their corresponding speed values. Hence, for a given speed, a function returns a coefficient vector $\boldsymbol{\alpha}^k$ in order to generate a walking motion with the desired speed as follows:

$$\boldsymbol{\theta}(s) = \boldsymbol{\theta}_0 + \boldsymbol{A}^{walk,k}(s)\mathbf{E}^{k^T} \tag{2.7}$$

We analyze the PCA spaces of N_{subj} subjects. For each PCA space $\mathbf{E}^k$ attached to a given subject k, the coefficient vectors $\boldsymbol{\alpha}^k$ of the $N_{seq}^{walk,k}$ original motions are compared to their corresponding speed values. Fig. 2.6 gathers the resulting comparison for the first two PC dimensions, for each subject in their own space. We determine a function model

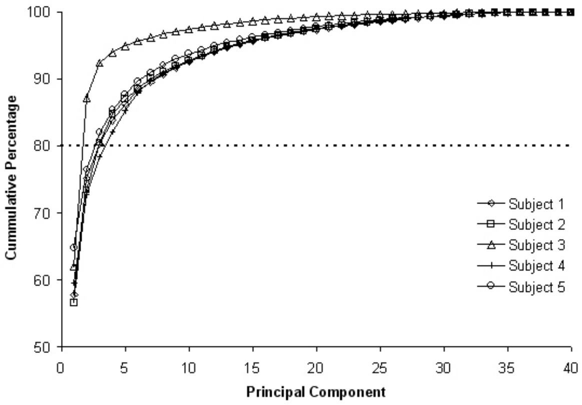


Figure 2.5: The cumulative percentage of the PCs for the five subjects of our database.

which best fits coefficient with speed values. At a first glance, the data seem to be fit by a polynomial model. However, such a model is unenforceable for extrapolation, in case motions are generated beyond their original speed range. Indeed, a parabolic approximation on the data of Fig. 2.6 (left) returns a same coefficient value α_1^k (and therefore a same motion $\boldsymbol{\theta}$) for two distinct speeds. For functions of higher degree number, undesired variations appear in-between the sampled data, leading also to a non-bijective function. However, a linear relationship is clearly apparent for motions having a speed greater than 3.0 km/h, representing the minimal captured speed.

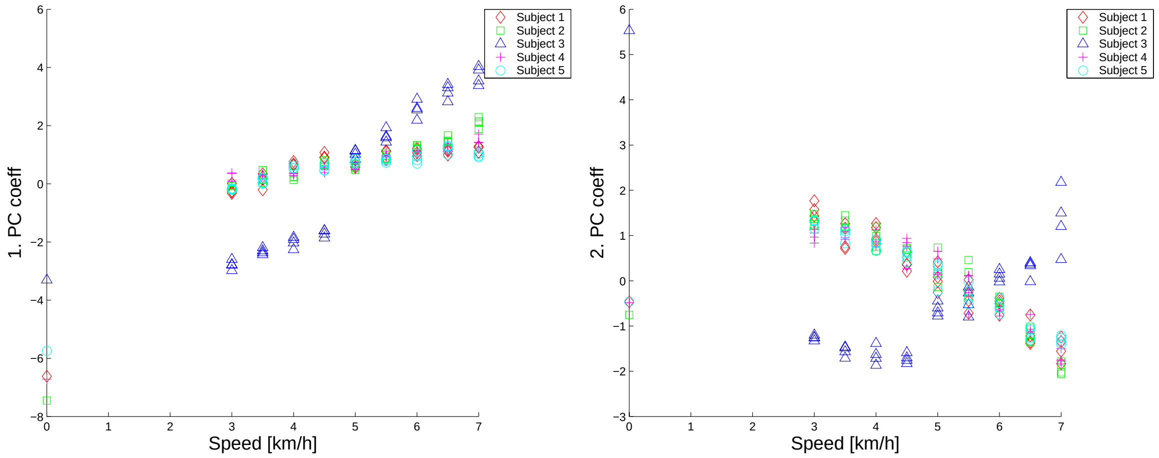


Figure 2.6: Comparison between the motion speed values and their corresponding coefficient for the five subjects. **Left:** The first principal coefficient values α_1^k. **Right:** The second principal coefficient values α_2^k.

Our proposed approach consists in describing a vector function $\boldsymbol{A}^{walk,k}(s)$ as follows:

$$\boldsymbol{A}^{walk,k}(s) = \begin{cases} \boldsymbol{A}_{lo}^{walk,k}(s) & \text{for } 0 \leq s < s_w \\ \boldsymbol{A}_{hi}^{walk,k}(s) & \text{for } s \geq s_w \end{cases} \tag{2.8}$$

where s_w corresponds to the minimal captured speed in the input motion.

The first function $\boldsymbol{A}_{hi}^{walk,k}(s)$ is computed by considering the coefficient vectors $\boldsymbol{\alpha}^k$ of the original motions and their corresponding speed values s, excluding the speed value zero. Hence $\boldsymbol{A}_{hi}^{walk,k}(s)$ approximates $\boldsymbol{\alpha}^k$ by $\widetilde{\boldsymbol{\alpha}}^k$ for a given speed s:

$$\boldsymbol{A}_{hi}^{walk,k}(s) = \widetilde{\boldsymbol{\alpha}}^k = \mathbf{a}s + \mathbf{b} \tag{2.9}$$

where $\mathbf{a}$ and $\mathbf{b}$ have to be determined. A linear least square fit is performed over the j increasing speed parameter values, by minimizing the sum of the square distances between the j-th coefficient vector $\boldsymbol{\alpha}_j^k$ with their associated speed s_j, and the approximated one $\widetilde{\boldsymbol{\alpha}_j}^k$, as the following equation describes:

$$\sum_{j=1}^{n} \left(\boldsymbol{\alpha}_j^k - \widetilde{\boldsymbol{\alpha}_j}^k\right)^2 = \sum_{j=1}^{n} \left(\boldsymbol{\alpha}_j^k - \mathbf{a}s_j - \mathbf{b}\right)^2 \tag{2.10}$$

where $n = N_{seq}^{walk,k} - 1$ is the number of motions of the subject k, excluding the standing posture having a null speed, represented by the coefficient $\boldsymbol{\alpha}_0^k$.

The approximation of the second vector function $\boldsymbol{A}_{lo}^{walk,k}(s)$ can also be performed by a linear model, between the coefficients $\boldsymbol{\alpha}_0^k$ and $\boldsymbol{A}_{lo}^{walk,k}(s_w)$. However, a linear interpolation between motion data, which is non-linear, produces unsatisfying animation results if the motion postures are rather different. As $\boldsymbol{\alpha}_0^k$ represents a standing and $\boldsymbol{A}_{lo}^{walk,k}(s_w)$ walking postures, it is therefore evident that the linear approximation is not well appropriate.

The idea is to give much more influence to the coefficient of the walking posture. We model a function which has a slope important at $\boldsymbol{A}_{lo}^{walk,k}(0)$, and near to zero at $\boldsymbol{A}_{lo}^{walk,k}(s_w)$. The behavior of the function $y = x^m$, plotted in Fig. 2.7 for $m = 0.2 \ldots 1$, corresponds to our need. The function slope can be adapted by varying m, from $m = 1$ to produce the previous linear solution to smaller values increasing the slope at $x = 0$.

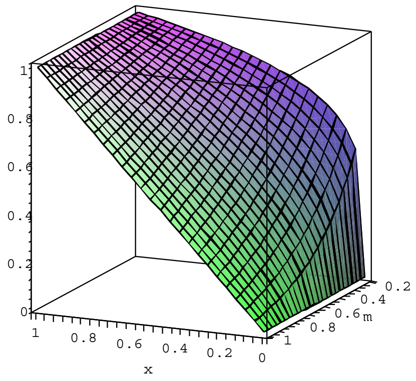


Figure 2.7: The function x^m for $x = 0 \ldots 1$ and $m = 0.2 \ldots 1$.

This model is adapted to our data, leading to the definition of the vector function $\boldsymbol{A}_{lo}^{walk,k}(s)$ as follows:

$$\boldsymbol{A}_{lo}^{walk,k}(s) = \boldsymbol{\alpha}_0^k + \mathrm{b}' s^m \tag{2.11}$$

where

$$\mathrm{b}' = \frac{\boldsymbol{A}_{lo}^{walk,k}(s_w) - \boldsymbol{\alpha}_0^k}{{s_w}^m} \tag{2.12}$$

In our experiments, we define $m = 0.4$.

To summarize this sub-section, a motion unit $\boldsymbol{\theta}(s)$ can be generated by varying the speed parameter s. For a given subject k, $\boldsymbol{A}^{walk,k}(s)$ defined in Eq. 2.8 returns a coefficient vector $\boldsymbol{\alpha}$ which is substituted in Eq. 2.6 to compute $\boldsymbol{\theta}(s)$, leading to the Eq. 2.7. The described approximation function $\boldsymbol{A}^{walk,k}(s)$ allows not only to interpolate, but also to extrapolate walking cycles beyond the captured domain.

We propose to extend this motion modeling for more high-level motion parameters. Besides walking, the running is the most used locomotion pattern. Moreover, it is necessary to add stylistic variations between produced animations. In the next sub-section, we extend our PCA-based modeling method to higher parameter dimensions and we aim at using the similar approximation functions we describe in this sub-section.

2.3.2 Multiple High-Level Parameter Variation

In addition to the speed, we aim to increase the parameter dimension of the motion vector $\boldsymbol{\theta}$. We therefore introduce more variety by constructing a motion matrix M composed of walking and running cycles from N_{subj} different subjects. In this configuration, three high-level parameters should drive the motion generation:

- The locomotion speed s
- The locomotion weight w_{loco} which varies between the two locomotion types walk ($w_{loco} = 0$) and run ($w_{loco} = 1$).
- The personification vector $\mathbf{w}_{subj}$ which contains N_{subj} elements. Each of those elements is defined by $w_{subj,k}$ and corresponds to the normalized weight assigned to the k-th subject so that $\sum_{k=1}^{N_{subj}} w_{subj,k} = 1$.

All of those build a parameter vector $\boldsymbol{\Psi} = (s, w_{loco}, \mathbf{w}_{subj})$ which allows the characterization of a locomotion cycle $\boldsymbol{\theta}(\boldsymbol{\Psi})$.

2.3.2.1 Main PCA Level

Similarly to the previous sub-section 2.3.1, a PCA algorithm is performed on the matrix $\widetilde{\mathrm{M}}$, by subtracting the mean motion $\boldsymbol{\theta}_0$ from M, producing a PCA space as described in Eq. 2.13. Note that M contains all motions from N_{subj} subjects, including the standing posture for each subject. We call this new space the ”**main PCA**”. Fig. 2.8 depicts all $\boldsymbol{\alpha}$ of the original walking, running and standing motions, in the first two PCs.

$$\boldsymbol{\theta} \approx \boldsymbol{\theta}_0 + \sum_{i=1}^{m} \alpha_i \mathbf{e}_i = \boldsymbol{\theta}_0 + \boldsymbol{\alpha}\mathbf{E}^T \tag{2.13}$$

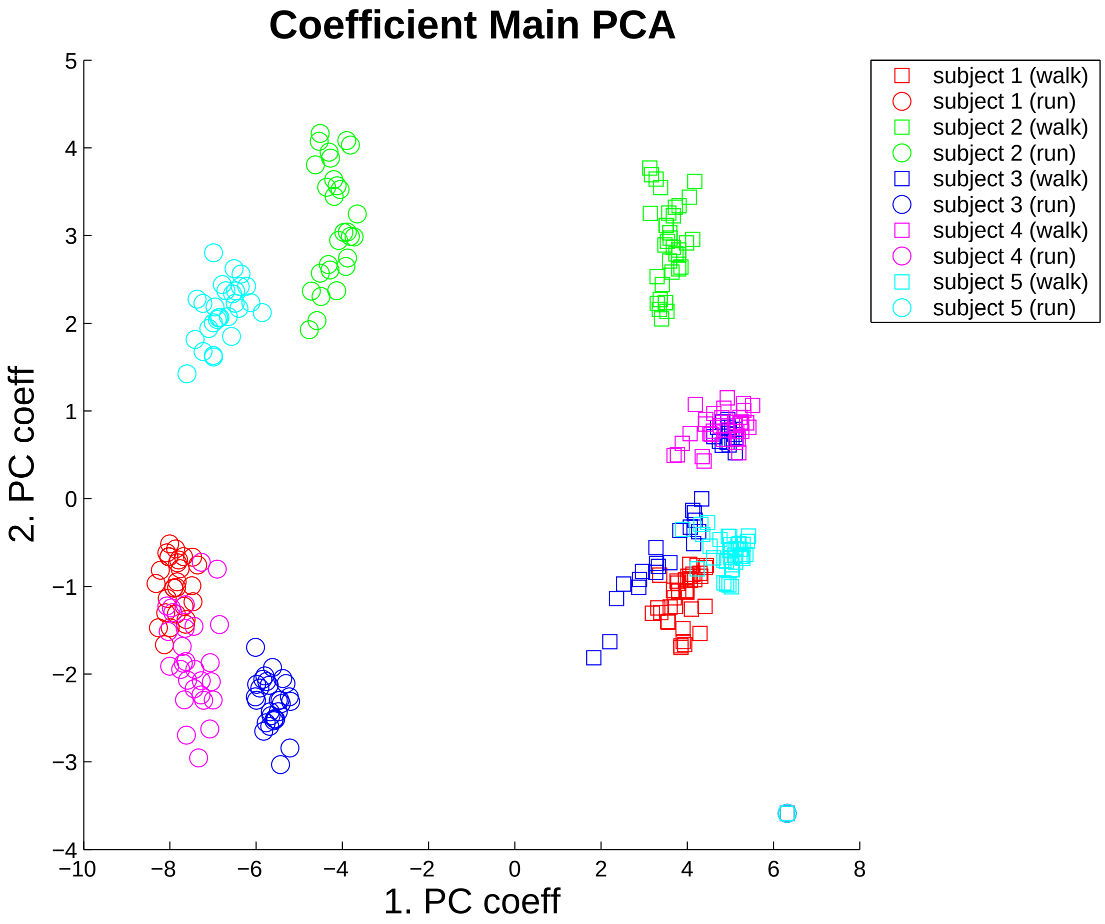


Figure 2.8: The motion database projected into the main PCA space (first two PCs). Observe the clustering behavior by subject and type of locomotion.

From this main PCA, we want on the one hand to improve the efficiency of the motion generation and on the other hand to perform correct speed extrapolation, as done before in case of walking motions from a single subject. These goals can be achieved by using RBF (Radial Basis Function) and polynomial functions to perform multidimensional scattered data interpolation, initially introduced by Rose and Cohen in [Rose et al., 1996]. To increase the efficiency of this method, an enhanced version is presented in [Sloan et al., 2001], and used for human locomotion generation by Park et al. [Park et al., 2002a]. The main idea of these techniques consists in associating weight values to the original motions, according to a given parameter vector. However, this method has a drawback: all original motions contribute to the computation of a new motion. Actually, even if the user desires a parameterized motion from a single subject, the weights of all other subject's motions may not be null. Therefore, this method is limited to a small number of original data, to limit the computation time. As soon as the speed parameter value, for a given subject, goes beyond the captured data, the resulting motions are not satisfying as they are, influenced by the weights of other subjects, as illustrated in Fig. 2.9.

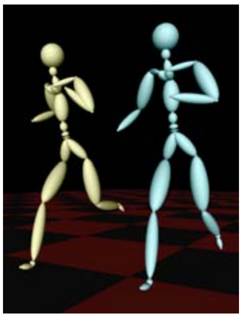


Figure 2.9: Method comparison for a running motion generated beyond capture data (16 [km/h]). The left character posture (yellow) is obtained by polynomial and RBF method and the right one (blue) by ours. Observe the difference at the elbow joint.

In concrete terms, we introduce a method that treats the speed extrapolation individually for each subject and type of locomotion. Firstly we aim at excluding the influence of other motions, as opposed to [Rose et al., 1998], and secondly we use the method presented in the previous sub-section (2.3.1) where only the speed parameter varies. At a first thought, the example motions in the main PCA (Fig. 2.8) can be classified by subjects and type of locomotion. Then a linear least square fitting may be applied on these groups of motions differing only by their locomotion speeds. Therefore, for a given speed, the resulting fitting function provides the coefficient combination of the main PCA to compute the desired motion. However, we propose a hierarchical PCA space structure that helps to classify the motions and allows to perform the linear least square in a lower dimension than the one of the main PCA space.

The main PCA, as illustrated in Fig. 2.8, shows relatively compact clusters related to subjects and type of locomotion. As the final goal is to group motion data having only one parameter that varies (the speed), the classification can be divided into two successive stages: the first which groups the data per subject, and then the second which subdivides theses groups by type of locomotion. This classification can be executed automatically by applying a simple clustering method like K-means [MacQueen, 1967]. For the fist stage we give the number of clusters (two in our case). Hence, the algorithm distinguishes exactly walking from running motions, as illustrated in Fig. 2.10, left. Then, for each of those clusters, we indicate to the algorithm one sample per subject to improve the classification accuracy. As a result, all running motions are separated by subject correctly, whereas only three walking motions are assigned to their corresponding subject incorrectly, as depicted in Fig. 2.10, right. This error has to be manually corrected.

At each of these two classification stages, the PCA algorithm is executed on each created group of motions. It results a set of PCA spaces which are arranged in order to build a hierarchical structure. The question is how to determine this structure. The first intuitive ideas consists in separating the motion data by their type of locomotion, as Fig. 2.8 suggests. Actually, the first component seems to isolate running from walking motions. However, two reasons turn this observation down. One concerns computation efficiency and the other the

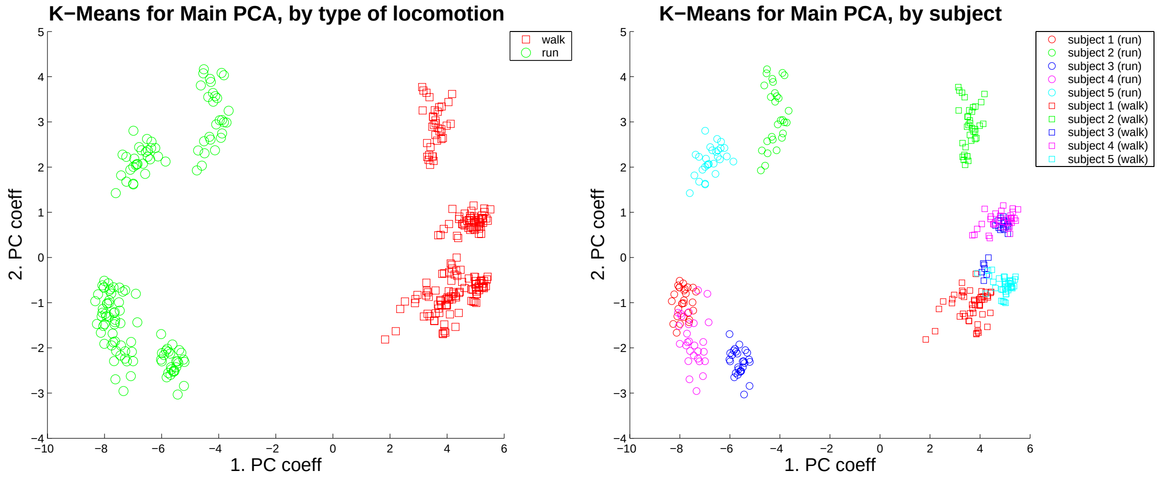


Figure 2.10: Classification results obtained with the K-means algorithm. **Left:** Classification by type of locomotion. **Right:** Classification by subject.

quality of the animation. Before going into details, we present our hierarchical method which separates the subjects before their type of locomotion. A complete overview of the hierarchy is shown in Fig. 2.11.

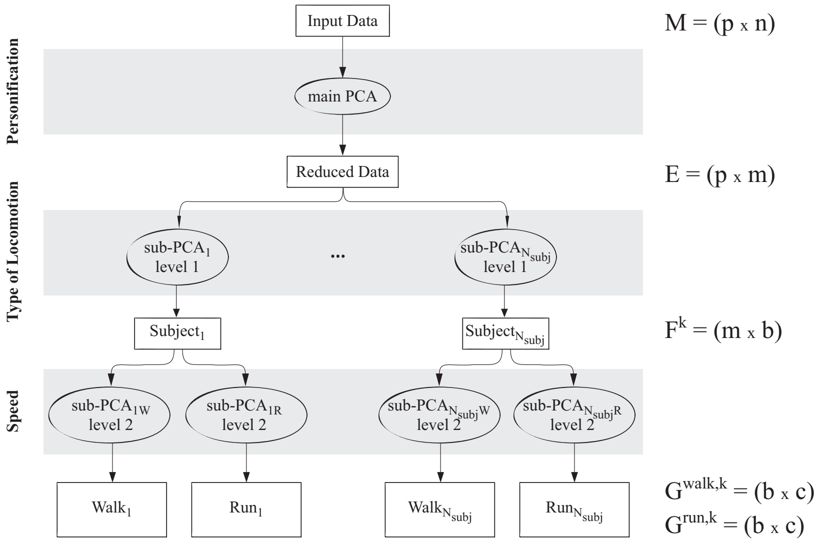


Figure 2.11: Overview of the hierarchical structure of PCA spaces. On the right, the matrices (with their dimensions) describing the space at each level (k stands for k-th subject).

2.3.2.2 Sub-PCA Level 1

The main PCA's coefficient vectors $\boldsymbol{\alpha}$ are grouped by subject and form N_{subj} groups. The group related to the subject k is composed of N^k_{seq} coefficient vectors. A second PCA algorithm is applied to those groups, resulting in N_{subj} new PCA spaces, referred to as "**sub-PCA level 1**". In one of these spaces, a coefficient vector $\boldsymbol{\alpha}^k$ relative to a specific subject k can be expressed as follows:

$$\boldsymbol{\alpha}^k \approx \boldsymbol{\alpha}_0^k + \sum_{j=1}^{b} \beta_j^k \mathbf{f}_j^k = \boldsymbol{\alpha}_0^k + \boldsymbol{\beta}^k \mathbf{F}^{k^T} \tag{2.14}$$

where $\boldsymbol{\beta}^k$ represent a coefficient vector and $\mathbf{F}^k = (\mathbf{f}_1^k, \mathbf{f}_2^k, \ldots, \mathbf{f}_b^k)$ the b first eigenvectors, with $b \leq N_{seq}^k$. The vector $\boldsymbol{\alpha}_0^k$ represents the average of the N_{seq}^k coefficient vectors $\boldsymbol{\alpha}^k$ for a subject k.

The first two dimensions of the $\boldsymbol{\beta}^k$, corresponding to all motions relative to a subject k expressed in $\mathbf{F}^k$, are plotted and superimposed for all subjects in Fig. 2.12. From this figure, three distinct clusters clearly emerge, one for each locomotion type (standing, walking and running). For all subjects (they do not share identical PCA spaces, but are superimposed for graphical representation convenience), the first PC separates data from walking and running, while the standing posture (represented by the outliers in Fig. 2.12) is more considered as a special case of walking. The visualization in the first two PCs is sufficient, as their eigenvalues explain the most significant variations among the $\boldsymbol{\alpha}^k$.

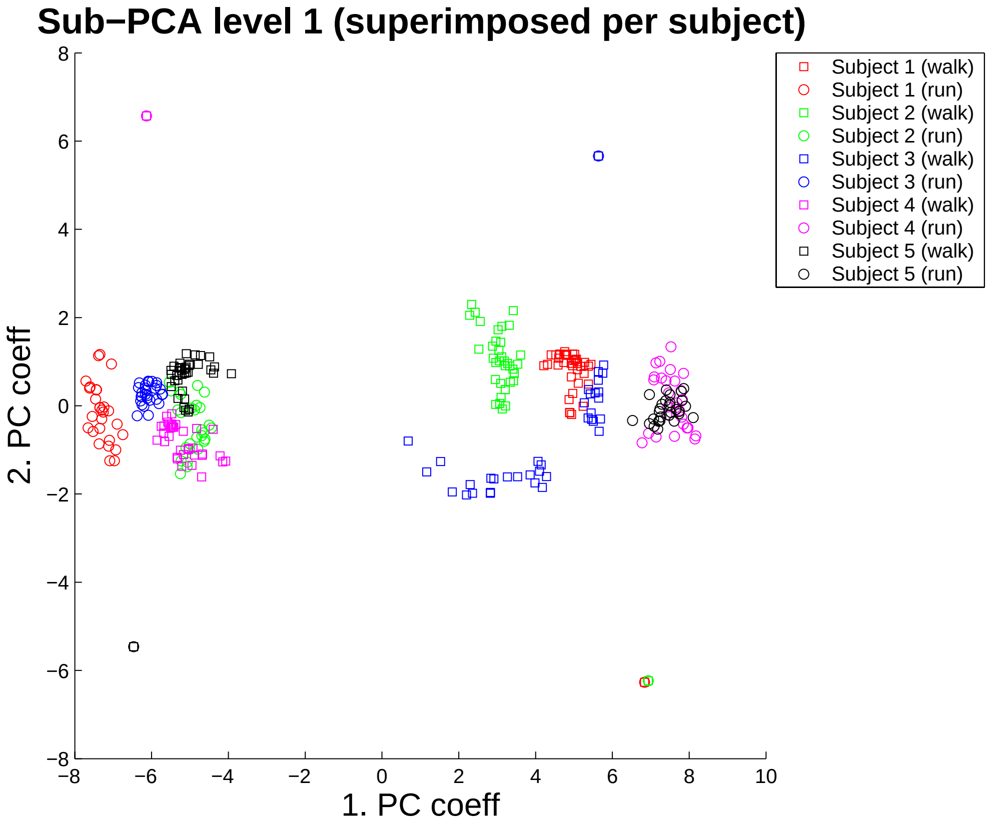


Figure 2.12: The fist two values of the coefficient vectors $\boldsymbol{\beta}^k$ for all k subjects (superimposed) in the sub-PCA level 1.

These sub-PCA level 1 spaces are well-suited to parameterize a motion with the second high-level parameter: the locomotion weight w_{loco}. In fact, in each $\mathbf{F}^k$, the coefficient $\boldsymbol{\beta}^k$

are classified in two groups, by locomotion type: walking and running. We therefore identify $N_{seq}^{walk,k}$ coefficient vectors for walking, referred to as $\boldsymbol{\beta}^{walk,k}$, and $N_{seq}^{run,k}$ for running, referred to as $\boldsymbol{\beta}^{run,k}$. Note that the coefficient vector representing the standing posture is integrated in both groups in order to give a lower bound (where the speed is null) for the generation of new motion units.

Hence, for a given subject k and given speed s, a linear interpolation is performed in the PCA space $\mathbf{F}^k$ between $\boldsymbol{\beta}^{walk,k}$ and $\boldsymbol{\beta}^{run,k}$. The resulting $\boldsymbol{\beta}^k$ is therefore obtained as follows:

$$\boldsymbol{\beta}^k = (1 - w_{loco})\boldsymbol{\beta}^{walk,k} + w_{loco}\boldsymbol{\beta}^{run,k} \tag{2.15}$$

2.3.2.3 Sub-PCA Level 2

However, the $\boldsymbol{\beta}^{walk,k}$ and $\boldsymbol{\beta}^{run,k}$ remains to be computed, according to the high-level parameter speed s. We construct the last level of our hierarchy by applying again a PCA algorithm on each group of the $\boldsymbol{\beta}^{walk,k}$ and $\boldsymbol{\beta}^{run,k}$ respectively. This operation performed on each subject, leads to $2N_{subj}$ new PCA spaces, called "**sub-PCA level 2**". For example, the PCA for walking motions of a given subject k is described as follows:

$$\boldsymbol{\beta}^{walk,k} \approx \boldsymbol{\beta}_0^{walk,k} + \sum_{j=1}^{c} \gamma_j^{walk,k} \mathbf{g}_j^{walk,k} = \boldsymbol{\beta}_0^{walk,k} + \boldsymbol{\gamma}^{walk,k} \mathbf{G}^{walk,k^T} \tag{2.16}$$

where $\boldsymbol{\gamma}^{walk,k}$ is the coefficient vector and $\mathbf{G}^{walk,k} = (\mathbf{g}_1^{walk,k}, \mathbf{g}_2^{walk,k}, \ldots, \mathbf{g}_c^{walk,k})$ the c first eigenvectors ($c \leq N_{seq}^{walk,k}$). The vector $\boldsymbol{\beta}_0^{walk,k}$ represents the average of all coefficient vectors $\boldsymbol{\beta}^{walk,k}$ for the k subject. The basis $\mathbf{G}^{run,k}$ for the running motions is analogously computed.

This third hierarchical level is composed of different PCA spaces having motions that differ only by their speed values, a similar situation as presented in the previous section (2.3.1). The coefficient vectors $\boldsymbol{\gamma}^{walk,k}$ and $\boldsymbol{\gamma}^{run,k}$ of each sub-PCA level 2 are depicted in Fig. 2.13 with respect to their corresponding speed. The spaces of each subject are superimposed, with the walking motions on the left side and running motions on the right side respectively of the figure.

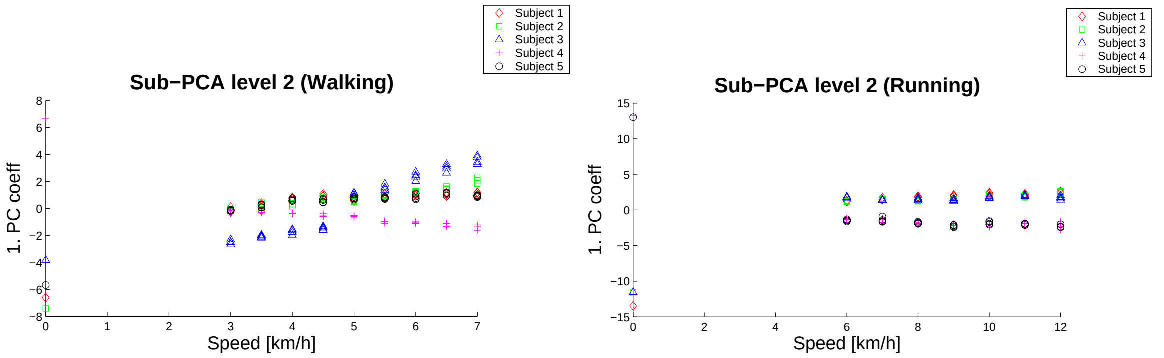


Figure 2.13: Speed and the first dimension of the coefficient vectors (sub-PCA level 2) for all subjects. **Left:** $\gamma^{walk,k}$ corresponding to walking. **Right:** $\gamma^{run,k}$ corresponding to running.

We apply therefore the same approximation method used for a single parameter variation. For each subject k, two approximation functions are defined similar to Eq. 2.9 and Eq. 2.11. Their goal is to determine a coefficient vector $\gamma^{walk,k}$ and $\gamma^{run,k}$ respectively for a given speed s. Those two functions are described as follows:

$$\boldsymbol{A}^{walk,k}(s) = \begin{cases} \boldsymbol{A}_{lo}^{walk,k}(s) & \text{for } 0 \leq s < s_w \\ \boldsymbol{A}_{hi}^{walk,k}(s) & \text{for } s \geq s_w \end{cases} \qquad \boldsymbol{A}^{run,k}(s) = \begin{cases} \boldsymbol{A}_{lo}^{run,k}(s) & \text{for } 0 \leq s < s_r \\ \boldsymbol{A}_{hi}^{run,k}(s) & \text{for } s \geq s_r \end{cases} \tag{2.17}$$

2.3.2.4 Motion Unit Computation

The computation of a motion unit with respect the parameter vector $\boldsymbol{\Psi}$ is computed in a bottom-up way, from the lowest to the highest hierarchy level. For each PCA level, one or more coefficient vectors are computed according to a specified parameter from $\boldsymbol{\Psi}$. In general, a motion unit $\boldsymbol{\theta}(\boldsymbol{\Psi})$ is computed as follows:

$$\boldsymbol{\theta}(\boldsymbol{\Psi}) = \boldsymbol{\theta}_0 + \left[\sum_{k=1}^{N_{subj}} w_{subj,k} \left(\boldsymbol{\alpha}_0^k + \boldsymbol{\beta}^k(\boldsymbol{\Psi}) \mathbf{F}^{k^T} \right) \right] \mathbf{E}^T \tag{2.18}$$

where $\boldsymbol{\beta}^k(\boldsymbol{\Psi})$ is described as:

$$\boldsymbol{\beta}^k(\boldsymbol{\Psi}) = (1 - w_{loco}) \left(\boldsymbol{\beta}_0^{walk,k} + \boldsymbol{A}^{walk,k}(s) \mathbf{G}^{walk,k^T} \right) + w_{loco} \left(\boldsymbol{\beta}_0^{run,k} + \boldsymbol{A}^{run,k}(s) \mathbf{G}^{run,k^T} \right) \tag{2.19}$$

The resulting vector functions associated to each lowest PCA space of the hierarchical structure allows to compute the corresponding coefficient vectors $\mathbf{G}^{walk,k}$ and $\mathbf{G}^{run,k}$ given the high-level parameter s. We show the resulting functions for only one subject ($k = 1$). The results for the others subjects are depicted in Appendix C, from Fig. C.1 to Fig. C.4.

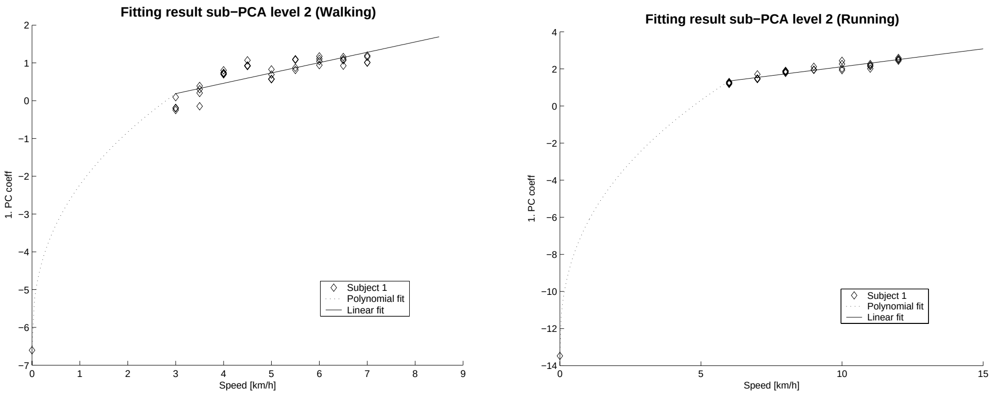


Figure 2.14: Resulting approximation functions for walking (left) and running (right) for the subject $k = 1$.

2.3.2.5 Hierarchical Order

As mentioned previously, it remains to explain the choice of separating the data first by subject. Imagine the situation with N_{subj} subjects who compose the database, having each of them $N_{seq}^{walk,k}$ walking and $N_{seq}^{walk,k}$ running motion samples. Independently of the separation scheme, the third level of hierarchy is composed of $2N_{subj}$ sub-PCA level 2, one for walking and one for running motion per subject. The computation time to generate motion in these spaces is similar for both alternatives. The second level of hierarchy is different according to the classification method. If it is by subject, N_{subj} sub-PCA level 1 spaces containing each $N_{seq}^{walk,k} + N_{seq}^{run,k}$ motions are created, otherwise two spaces: one having N_{seq}^{walk} motions, the other N_{seq}^{run}, defined as follows:

$$N_{seq}^{walk} = \sum_{k=1}^{N_{subj}} N_{seq}^{walk,k} \tag{2.20}$$

$$N_{seq}^{run} = \sum_{k=1}^{N_{subj}} N_{seq}^{run,k} \tag{2.21}$$

Finally, the main PCA space is the same for both separation options, containing $N_{seq}^{tot} = N_{seq}^{walk} + N_{seq}^{run}$ motions.

The underlying idea is to perform the most often occurring interpolation in the lowest PCA space dimension. The separation first by type of locomotion is preferred when N_{subj} is significant or when all $w_{subj,k}$ contribute to the motion computation. On the contrary, when w_{loco} vary more often and few $w_{subj,k}$ contribute to the computation, the separation by subject is preferred. We believe that the parameter vector $\mathbf{\Psi}$ which defines a locomotion style is less used than the parameter w_{loco} allowing transition from walk to run. In addition to this computation time optimization, the prior separation by subjects produces better results in practice, especially when extrapolating cycles beyond captured data.

2.4 Jump Motion Units

We can apply the motion interpolation and extrapolation method presented in the former section to other classes of motion. The genericity potential of the method is demonstrated by applying it to long jump motion database described in the previous chapter 1, section 1.1. These motions have a different structure than the locomotion: they are non-cyclic and have not been recorded on a treadmill. Moreover, other parameters are varying: the type and length of the jump.

Similar to the locomotion case, a motion matrix $\mathbf{M}$ is composed of jumping sequences having three parameter variations: the subject, the type of jump (walking or running run-up) and jump length. The standing posture is not added to $\mathbf{M}$. The reason is that the minimal jump length is greater than zero, and near to the minimal jump length the subject was asked to perform. The goal is to compute a jump $\boldsymbol{\theta}(\mathbf{\Psi})$ where the jump length l is substituted for the speed s in $\mathbf{\Psi}$.

PCA is applied on the derived matrix $\widetilde{\mathrm{M}}$ and the resulting main PCA space (computed as described in Eq. 2.13) is depicted in Fig. 2.15. The hierarchical PCA method is applied to separate the motions first by subjects, then by type of jump and finally by jump length. In this last hierarchical level, a linear least square fit is performed between the motion coefficients and their corresponding jump length values.

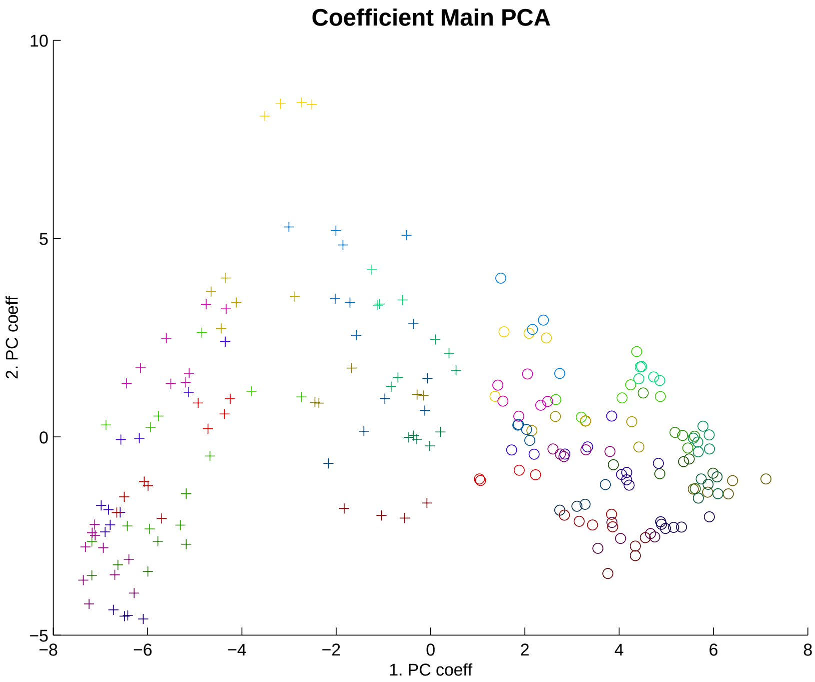


Figure 2.15: Walking jumps (cross) and running jump (circle) in the first two PCs. A color, whose intensity is proportional to the jump length, is assigned to each subject (totally 7 subjects).

We improve the efficiency of the jumping computation by treating separately the walking and running run-up jumps. Actually, the parameter w_{loco} is either 0 or 1, but never in-between, as opposed to the locomotion where smooth transition from walking to running (or inversely) may occur. Hence two independent jump motion structures are created, leading to a hierarchical structure of only two levels: the main PCA in which the parameter vector $\mathbf{w}_{subj}$ drive the interpolation and the sub-PCA level 1 (described in Eq. 2.14) where the parameter l is used to compute corresponding coefficient vectors. The parameter vector $\boldsymbol{\Psi}$ is therefore simplified by removing the w_{loco} parameter.

At this latter level, a vector function $\boldsymbol{A}^{jmpW,k}(l)$ (in case of walking run-up jumps) and $\boldsymbol{A}^{jmpR,k}(l)$ (in case of running run-up jumps) is necessary for each space attached to each subject k. This function is composed of a single linear approximation. The Fig. 2.17 illustrates the relation between the walking (left) and running (right) jump length and the first dimension of their coefficients for all subjects. Note that for graphical representation convenience, the sub-PCA level 1 spaces are superimposed. The functions $\boldsymbol{A}^{jmpW,k}(l)$ and $\boldsymbol{A}^{jmpW,k}(l)$ are computed similarly to Eq. 2.9 where the coefficient vectors $\boldsymbol{\beta}^k$ are substituted for $\boldsymbol{\alpha}$.

For the subject $k = 1$, the results in the sub-PCA level 1 spaces for the first PCs are illustrated in Fig. 2.16. We consider also the other remaining PCs, as most of the case at least two PCs are needed. Here, the linear fit is less accurate. In fact, as those PCs are less influential, the data are more spread. The results for the others subjects are depicted in Appendix C, from Fig. C.5 to Fig. C.8.

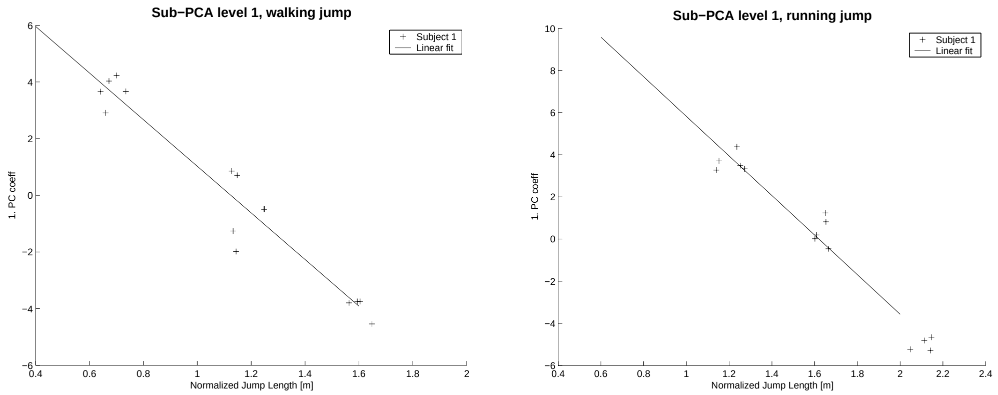


Figure 2.16: Resulting $\boldsymbol{A}^{jmpW,k}(l)$ function for walking (left) and $\boldsymbol{A}^{jmpR,k}(l)$ for running (right) jump for the subject 1.

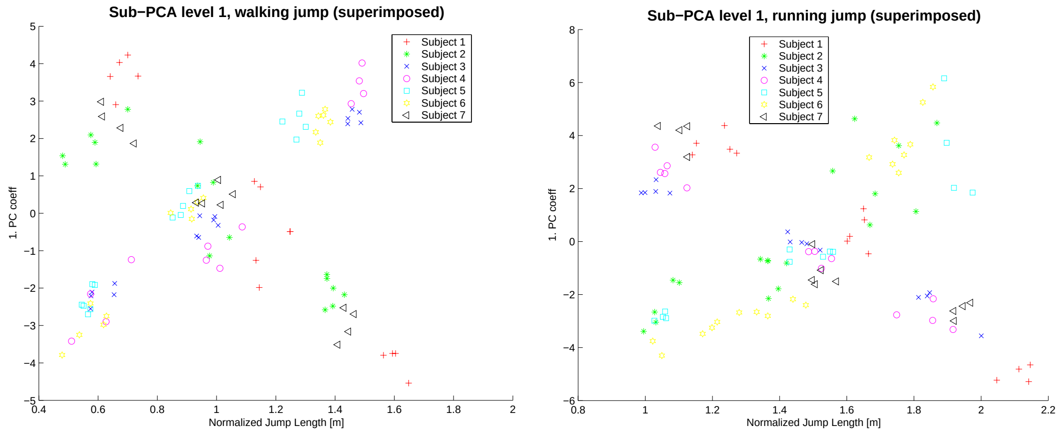


Figure 2.17: Jump length and the first coefficient vectors β for all subject (left walking jump, right running jump).

To summarize, a walking jump $\boldsymbol{\theta}(\boldsymbol{\Psi})$ is computed as follows:

$$\boldsymbol{\theta}(\boldsymbol{\Psi}) = \boldsymbol{\theta}_0 + \left[\sum_{k=1}^{N_{subj}} w_{subj,k}\left(\boldsymbol{\alpha}_0^k + \boldsymbol{A}^{jmpW,k}(l)\mathbf{F}^{k^T}\right)\right]\mathbf{E}^T \tag{2.22}$$

A running jump is computed analogously.

2.5 Results

The presented motion modeling method structures motion capture data in order to generate new locomotion cycles or jump sequences. In this section, we describe precisely the construction of the PCA hierarchical structure.

From five captured subjects (see Table 2.1) having performed locomotion activities, four cycles have been extracted for each speed ranging from 3 to 7 [km/h] by increment of 0.5 [km/h] for walking, and from 6 to 12 [km/h] by increments of 1 [km/h] for running. In the end, it leads to a total of 300 motion units, sampled and time-normalized arbitrarily at 25 frames, and represents an amount of 5 MB of data storage.

Activity	**Subject Identification**	**Gender**	**Leg length (in meters)**
Locomotion	1	Male	0.93
	2	Female	0.88
	3	Male	0.98
	4	Female	0.91
	5	Male	0.87
Jump	1	Male	0.86
	2	Male	1.0
	3	Male	0.98
	4	Female	0.88
	5	Male	0.98
	6	Female	0.90
	7	Male	0.87

Table 2.1: Description of the subjects in our locomotion and jump databases.

The hierarchical PCA structure applied to this database is described in Table 2.2. It shows the number of PCs and the percentage of original information they can retrieve. The data reduction factor of each hierarchical level is also indicated. From the original 5 MB of data only 200 kB is needed to store the complete hierarchical structure.

Name of PCA space	**Number of PCs**	**Data percentage**	**Data reduction factor**
Main PCA (**E**)	$m = 9$	95%	29
Sub-PCA level 1 ($\mathbf{F}^k$)	$b = 2$	95%	11
Sub-PCA level 2 ($\mathbf{G}^{walk,k}$)	$c = 1$	95%	3
Sub-PCA level 2 ($\mathbf{G}^{run,k}$)	$c = 1$	95%	3

Table 2.2: Description of PCA spaces structure for locomotion.

For the jumps, seven subjects (see Table 2.1) have performed totally 208 walking and running jumps whose distribution is summarized in the Table 2.3. We obtain approximately three different jump length for each type of jump. The distances range from 0.4 to 1.2 [m], by increments of 0.4 [m] for the walking jumps, respectively from 0.8 to 1.6 [m] for running jumps. Normalized to 25 frames, these sequences correspond to 3 MB.

The next Table 2.4 describes the two hierarchical structures, one for each type of jump. For the sub-PCA level 1 spaces, we select the minimal number of PCs that retain at least the 80% of the original data. Discrepancies are observed, mainly caused by the difficulty to perform many jumps with a similar style, on the contrary to locomotion cycles performed on a treadmill. In addition, the reduction factor is 10 for both main PCA (walking and running jumps) and approximately 21 for all other sub-PCA level 1.

Subject k	**#walking jumps**	**#running jumps**
1	15	14
2	17	18
3	16	14
4	12	13
5	16	13
6	16	18
7	13	13

Table 2.3: Number of jumps per subject.

PCA space name	**#PCs**	**Data percentage**
Main PCA (walking jump, $\mathbf{E}$)	$m = 9$	80%
Sub-PCA level 1 ($\mathbf{F}^1$)	$b = 1$	80%
Sub-PCA level 1 ($\mathbf{F}^2$)	$b = 3$	85%
Sub-PCA level 1 ($\mathbf{F}^3$)	$b = 1$	85%
Sub-PCA level 1 ($\mathbf{F}^4$)	$b = 2$	95%
Sub-PCA level 1 ($\mathbf{F}^5$)	$b = 2$	90%
Sub-PCA level 1 ($\mathbf{F}^6$)	$b = 2$	95%
Sub-PCA level 1 ($\mathbf{F}^7$)	$b = 1$	80%
Main PCA (running jump, $\mathbf{E}$)	$m = 9$	80%
Sub-PCA level 1 ($\mathbf{F}^1$)	$b = 2$	87%
Sub-PCA level 1 ($\mathbf{F}^2$)	$b = 2$	80%
Sub-PCA level 1 ($\mathbf{F}^3$)	$b = 2$	86%
Sub-PCA level 1 ($\mathbf{F}^4$)	$b = 2$	84%
Sub-PCA level 1 ($\mathbf{F}^5$)	$b = 2$	94%
Sub-PCA level 1 ($\mathbf{F}^6$)	$b = 1$	90%
Sub-PCA level 1 ($\mathbf{F}^7$)	$b = 2$	92%

Table 2.4: Description of PCA spaces structure for jump motion.

2.6 Conclusion

In this chapter, we have presented a method based on PCA to interpolate and extrapolate motion units. The representation of these non linear data is carefully chosen. A motion considered as sequences of 3D positions is unreliable, entailing the violation of basic physical properties. The constant length of the character's limb is no more preserved and the feet penetrate significantly in the ground. We therefore opt for a representation of joint state in the angle space. In order to apply motion data expressed in this non-linear space to the PCA algorithm, we use the exponential map parameterization. This is possible as our exponential maps contain either small angular values or small divergences between their axes of rotation.

Our method considers locomotion cycles or jump sequences as motion units, characterized by different parameters: personification (i.e. the performer's style), locomotion weight and speed (or jump length). The projection of these motion units into PCA spaces, structured hierarchically, allows to interpolate and also extrapolate new cycles or sequences given a parameter vector. As opposed to approaches computing a new motion frame by frame, our method generates at once the cycle or the complete sequence. Moreover, the successive motion classifications by specific characteristics improve the motion extrapolation quality.

In the next part of this thesis, we present results of this modeling method applied to 3D character animation. In fact, this presented method is not sufficient to animate characters by continuously varying the motions parameters. First, a normalization stage is necessary, in space as well as in time. The space normalization ensures the genericity of computed motions to any character size. As the input data are time-normalized, the duration of a motion has to be determined with respect to its parameters. Then a method is required to generate a continuous stream of motion with parameter variation. Finally, this motion modeling method is validated by comparing it with the original captured motions and is balanced against another possible alternative.

In addition, this stream can be improved by enforcing basic foot constraint, allowing straight locomotion to be turned into curved locomotion for example. Last but not least, the locomotion has to be smoothly harmonized with jump sequences to produce seamless transition. Hence, the next part of this thesis tackles also these requirements, essential for character animation, and provides incrementally method results applied to 3D character animation.

Part III

Character Animation

Chapter 1

Locomotion and Jump Engines

1.1 Introduction

An animation engine has to produce a continuous stream of motions driven by high-level parameters. This chapter presents the elaboration of two engines, one for the locomotion and the other dedicated to the jump activity. From the motion modeling method presented in the previous chapter, we propose to generalize the computed motion units in order to adapt them to any character's size and to reconstruct their timing. A character is then continuously animated by computing new motions inside the hierarchical PCA structure driven by high-level parameters. The resulting synthesized animations are then compared with the original captured motions.

The generalization of the used heterogeneous input data is an important aspect to produce animation adaptable to any kind of virtual humans. Indeed we captured various subjects having not only differences in motion parameter (speed, type of locomotion, jump length, type of jump), but also, and more significantly, differences in size. Two normalization stages are necessary: one in space, the other in time.

1.2 Space Normalization

The space normalization is performed on the input motion data, composed of translations and rotations. All translation values from the motion vectors $\boldsymbol{\theta}$, being in fact the position of the humanoid root $\widehat{\mathrm{p}}_r$, have to be divided by the leg length H of the captured subject. In our case, as all of our captured data are converted to a single character model, we divide $\widehat{\mathrm{p}}_r$ by the leg length H_{mocap} of this model. The rotation values do not need to be normalized, as explained in [Murray, 1967]. In this work, Murray demonstrated that all the leg relative angles in the sagittal plane (hip, knee and ankle) show very similar trajectories for all adult men for the same value of normalized speed s_n defined in Eq. 1.1.

$$s_n = \frac{s}{H} \tag{1.1}$$

We generalize this statement to the running. For jumping motions, the jump length is substituted for the locomotion speed and then normalized according to the leg length.

1.3 Time Normalization

The time normalization stage (or time-warping) is more complex. Every original input motion has a specific duration. However, as the motion matrix applied to PCA must have the same number of lines for each column, the motions have been time normalized. Consequently, every motion has an identical duration, i.e an identical number of frames. After having computed a new motion, the specific duration of this new motion has to be retrieved.

The standard time-warping methods [Bruderlin and Williams, 1995; Guo and Robergé, 1996; Rose et al., 1998; Park et al., 2002a; Kovar et al., 2002a] need to align explicitly the similar structures from the input motions. For example, when a transition is performed from a slow walking to a fast running motion, those methods vary continuously a normalized weight from 0 (i.e slow walking) to 1 (i.e fast running). The interpolation is therefore applied on two very different motion structures.

Our proposed method skips this alignment operation thanks to a subdivision interpolation scheme. For the previous example, it consists in interpolating successively between a walking and a running motion at a same given speed. The speed varies iteratively, from slow to fast, as well as the type of locomotion parameter (from walk to run). This simplification is reasonable since our motion database offers a big density of motions sampled by various speed values. For jumping motions, we observe that the essential motion information is described in one plane, namely the sagittal plane. Artifacts are therefore restricted to possible slight foot sliding. We introduce later in this thesis a method to correct those eventual motion shortcomings.

Our time-warping method is therefore limited to the construction of a function returning the corresponding generic time given the actual time of the animation. We define similarly to [Boulic et al., 1990, 2004], the generic (or normalized) time with a variable $\varphi = [0 \ldots 1]$, referred to as the cycle phase. The right heel strike event corresponds to the first frame of the cycle ($\varphi = 0$), the left heel strike to the middle frame of the cycle ($\varphi = 0.5$). At each time step $\Delta t = t_i - t_{i-1}$ of the actual time, the phase is updated according to the current motion parameter vector $\mathbf{\Psi}$ as follows:

$$\varphi_i = \varphi_{i-1} + \Delta\varphi = \varphi_i + \frac{\Delta t}{CycleDuration} = \varphi_i + \Delta t F(\mathbf{\Psi}) \pmod 1 \tag{1.2}$$

where $F(\mathbf{\Psi})$ is the frequency function. It allows to determine the cycle frequency of the current motion which is characterized by $\mathbf{\Psi}$.

In the next two sub-sections, we present a model to define the frequency functions F_{loco} for walking and running, and F_{jump} for jumping respectively.

1.3.1 Walking and Running Frequencies

The goal of the frequency function F_{loco} is to determine the frequency of the motion unit (walking or running cycle) for a given parameter vector $\boldsymbol{\Psi}$. To simplify, we start first to determine F_{loco} with respect to a single parameter: the normalized speed s_n. Inman in [Inman et al., 1981] proposes a model for this frequency function, focusing on walking patterns of various subjects. This model is called the Inman law and it is described in Eq. 1.3.

$$F_{loco}(s_n) = 0.743\sqrt{s_n} \tag{1.3}$$

We compare this function to our original data, by extracting the inverse duration of the performed cycles for each k subject and type of locomotion. Then their corresponding speeds are normalized to the leg length of the captured subject. Fig. 1.1 illustrates the evolution of the frequencies with respect to normalized speed for five subjects, and the comparison with the Inman law.

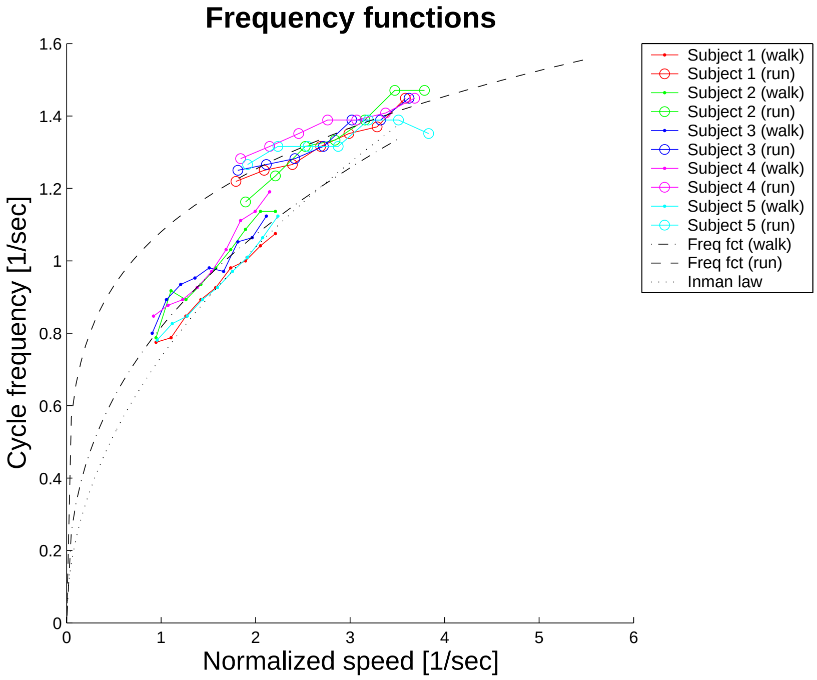


Figure 1.1: Comparison between the Inman law and the approximation of frequency functions for walking and running motions from our database

As this Inman law does not fit exactly our walking data and is does not apply to running data, we construct two new frequency functions for walking (F_{loco}^{walk}) and running (F_{loco}^{run}) motions respectively. Before describing them, two requirements are taken into account. First, a frequency function has to be defined per subject, as described in [Bruderlin and Calvert, 1996]. The authors mention that different subjects have a different ratio stride frequency versus stride length. Actually, each human adapts this ratio in order to get its most comfortable walk. The second requirement concerns the precision of the frequency function. In fact, a bad frequency induces a cycle frequency unsynchronized with the speed, producing foot sliding.

Hence, we define for each k-th subject a walking and a running frequency function respectively $F_{loco}^{walk,k}$ and $F_{loco}^{run,k}$. The accuracy of those functions is enhanced by an approach which computes the frequency based on synthesized instead of original motions. The frequency of a synthesized motion, defined as the inverse of the motion duration t_{motion}, is *a priori* unknown. However, it can be induced by measuring the path (or the traveled distance) p_{root} of the root joint $\widehat{\mathbf{p}}_r$ throughout a motion unit, for a given normalized speed s_n. Hence, the frequency function $F_{loco}^{walk,k}$ for a subject k can be computed as follows:

$$F_{loco}^{walk,k}(s_n) = \frac{1}{t_{motion}} = \frac{s_n}{p_{root}} \tag{1.4}$$

It remains to explain the computation of p_{root}. As the motions have been captured on a treadmill, their path can not be directly measured. We distinguish therefore two approaches, one for walking and one for running.

For walking motions, the path of a given motion can be approximated by doubling the step (or stride) length l_{step} measured in the first posture of this motion. The step length is the Euclidean distance measured between the right and the left heel joint, projected into the sagittal plane. However, instead of measuring this step length during the animation generation (i.e on-the-fly), we propose the construction of a function $f_{path}^{walk,k}(s_n)$. This function returns on-the-fly the doubled step length corresponding to a given speed s_n

According to the observations in [Bruderlin and Calvert, 1996], the stride length is linearly dependent on the speed. In our experiments, performed on motions generated by our modeling method, Fig. 1.2 shows a strong linearity between the speed and the stride length, starting from the minimal normalized speed $s_{n,min}^{walk}$ we captured. A linear least square is therefore computed between the measured step lengths and their corresponding speeds to determine a linear function $as_n + b$. Below $s_{n,min}^{walk}$, a polynomial approximation of type $c{s_n}^m$ is performed, where $m = 0.4$ yields experimentally to the best results. Hence, a function $f_{path}^{walk,k}(s_n)$ is determined as follows, for each subject k:

$$p_{root} \approx 2l_{step} = 2f_{path}^{walk,k}(s_n) = 2\begin{cases} c^{walk,k}{s_n}^{0.4} & \text{for } 0 < s_n \leq s_{n,min}^{walk} \\ a^{walk,k}s_n + b^{walk,k} & \text{for } s_n > s_{n,min}^{walk} \end{cases} \tag{1.5}$$

where $a^{walk,k}$ and $b^{walk,k}$ are the resulting coefficient of the linear least square performed on the subject k, and $c^{walk,k}$ the coefficient so that $c^{walk,k}{s_{n,min}^{walk}}^{0.4} = a^{walk,k}s_{n,min}^{walk} + b^{walk,k}$.

For running motions, the computation of the frequency function $F_{loco}^{run}(s_n)$ is carried out differently, as the stride length can not be directly extracted at the first posture of motion unit, due to the flying phase. In addition, as our data have been captured on a treadmill, the last posture of a motion unit does not hold translational information. The idea is therefore to scale the measured distance between right and left heel at the first frame by a factor r to obtain the correct l_{step}. By expermients, $r = 3$ produces the best results to limit foot sliding, for all captured subjects. The frequency function $F_{loco}^{run,k}$ is determined as follows, for the subject k:

$$F_{loco}^{run,k}(s_n) = \frac{s_n}{p_{root}} \tag{1.6}$$

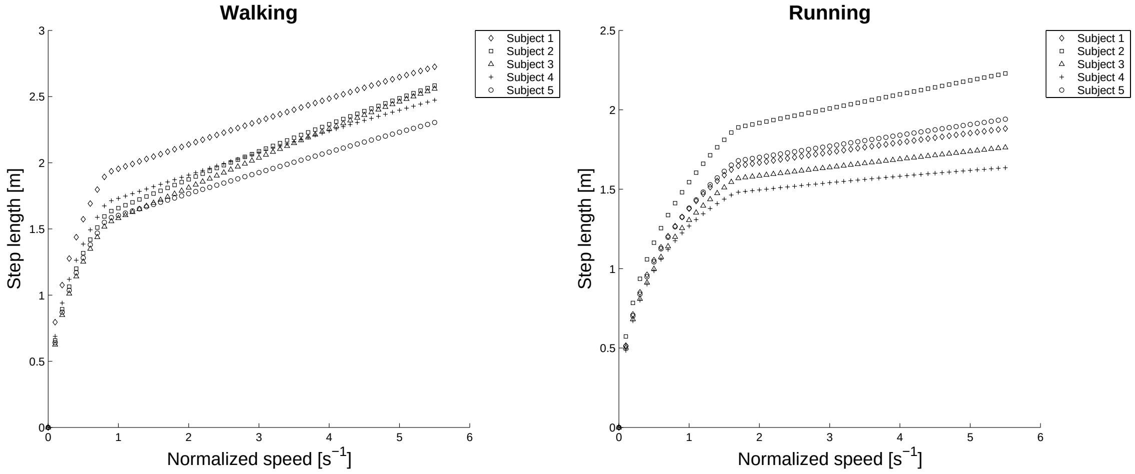


Figure 1.2: Stride length l_{step} given a normalized speed, for five subjects. **Left:** Walking cycles. **Left:** Running cycles.

where p_{root} is described as:

$$p_{root} \approx 3l_{step} = 3f_{path}^{run,k}(s_n) = 3 \begin{cases} c^{run,k} {s_n}^{0.4} & \text{for } 0 < s_n \leq s_{n,min}^{run} \\ a^{run,k} s_n + b^{run,k} & \text{for } s_n > s_{n,min}^{run} \end{cases} \tag{1.7}$$

The generalization of this time-warping function to the other high-level parameter variation (w_{loco} and $\mathbf{w}_{subj}$) is performed by a weighted linear interpolation. First, for each subject k, a frequency function $F_{loco}^{k}(s_n)$ is computed using a linear interpolation between the walking $F_{loco}^{walk,k}(s_n)$ and running $F_{loco}^{run,k}(s_n)$ function, according to the given locomotion weight w_{loco} parameter. Then the final frequency function is obtained by interpolating the $F_{loco}^{k}(s_n)$, weighted by the personification vector $\mathbf{w}_{subj}$. The final locomotion frequency is therefore defined as:

$$F_{loco}(\mathbf{\Psi}) = \sum_{k=0}^{N_{subj}} w_{subj,k} \Big[(1 - w_{loco}) F_{loco}^{walk,k}(s_n) + w_{loco} F_{loco}^{run,k}(s_n) \Big] \tag{1.8}$$

1.3.2 Jump Frequency

The time normalization for the jumping motions is quite different. On the contrary to cyclic motions whose duration variation is very slight between cycles having similar parameters, two original jumps may have a different duration for a given subject and jump length. In fact, the observation of our jumping data do not shows obvious relation between the jump length and its duration, as illustrated in Fig. 1.3. In practice, for a given subject and jump length, our motion model generates a jump as an approximation of all original jumps having the same length. The reason comes from the linear approximation performed on the PCA spaces. However, these original jumps have different duration and their influence on the generated jump is unknown. It is then difficult to determine exactly the duration of this new jump.

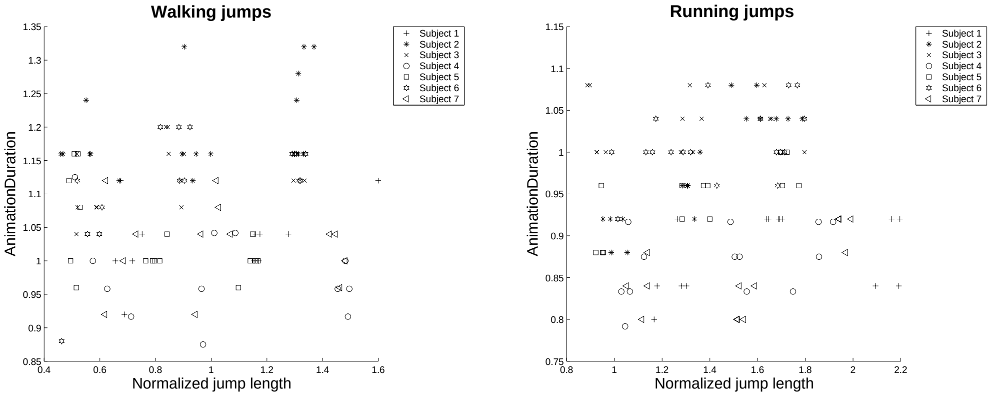


Figure 1.3: Comparison between the jump length and its duration, for seven subjects. **Left:** Walking jumps. **Right:** Running jumps.

Our goal is to find a frequency function F_{jump} which returns a unique frequency (and therefore a unique duration) for a given jump motion unit described by the parameter vector $\mathbf{\Psi}$. From a physical point of view, there is no direct relation between the jump length and its duration. In fact, the duration is linearly dependent only on the vertical jump speed measured at the end of the take-off phase, while the jump length depends also from the horizontal jump speed. In addition, our jump motion unit includes the take-off and landing phase whose durations have to be also computed. Therefore, we can not compute the jump duration purely physically.

For those reasons, we propose an approach similar to the locomotion case, by approximating the duration t_{motion} of a generated jump. To simplify, we consider first walking jumps with a variable normalized length l_n performed by a subject k. The frequency function is defined as:

$$F_{jump}^{walk,k}(l_n) = \frac{1}{t_{motion}} \approx \frac{\bar{s}}{p_{root}} \tag{1.9}$$

where $\bar{s}$ is the normalized average speed of the root joint during the jump. On the contrary to the locomotion case, this speed is not a measured parameter and is *a priori* unknown. We therefore elaborate a function $f_{\bar{s}}^{walk,k}$ which approximates $\bar{s}$ with respect to the known parameter l_n.

We analyze the original jumps to elaborate a relationship between the normalized mean speed $\bar{s}$ and jump length l_n. For a given original jump j, its speed $\bar{s}_j$ can be computed as follows:

$$\bar{s}_j = \frac{1}{N_j H} \sum_{i=2}^{N_j} \frac{||\widehat{\mathbf{p}}_{ri} - \widehat{\mathbf{p}}_{ri-1}||}{\Delta t_j} \tag{1.10}$$

where N_j is the frame number and Δt_j the frame rate of the original jump j, and H the subject's leg length. In order to determine the type of the approximation function $f_{\bar{s}}^{walk,k}(l_n)$, the correlation coefficients between the measured $\bar{s}$ and their associated normalized jump length are computed and summarized in Table 1.1. It results that the coefficient values are near to 1, indicating a strong linear relation. We can therefore apply a linear least square fit on the couples $\bar{s}$ and l_n to determine $f_{\bar{s}}^{walk,k}(l_n)$. For the running jumps, the similar approach is used to determine the function $f_{\bar{s}}^{run,k}(l_n)$.

Subject k	Walking jump	Running jump
1	0.9857	0.9345
2	0.9478	0.9149
3	0.9823	0.9603
4	0.9833	0.9577
5	0.9714	0.9617
6	0.9684	0.8841
7	0.9509	0.9486

Table 1.1: The correlation coefficients computed between the normalized jump mean speed $\bar{s}$ and its normalized jump length l_n, for a given subject and type of jump.

From Eq. 1.9, it remains to compute the path p_{root} of the root joint. For the similar reason than the locomotion case, we propose to compute this path on the synthesized motions. As the jumping motions are not capture on a treadmill, we can use their translation $\widehat{\mathbf{p}}_{ri}$ directly for the path computation:

$$p_{root} = \sum_{i=2}^{N_{frame}} ||\widehat{\mathbf{p}}_{ri} - \widehat{\mathbf{p}}_{ri-1}|| \tag{1.11}$$

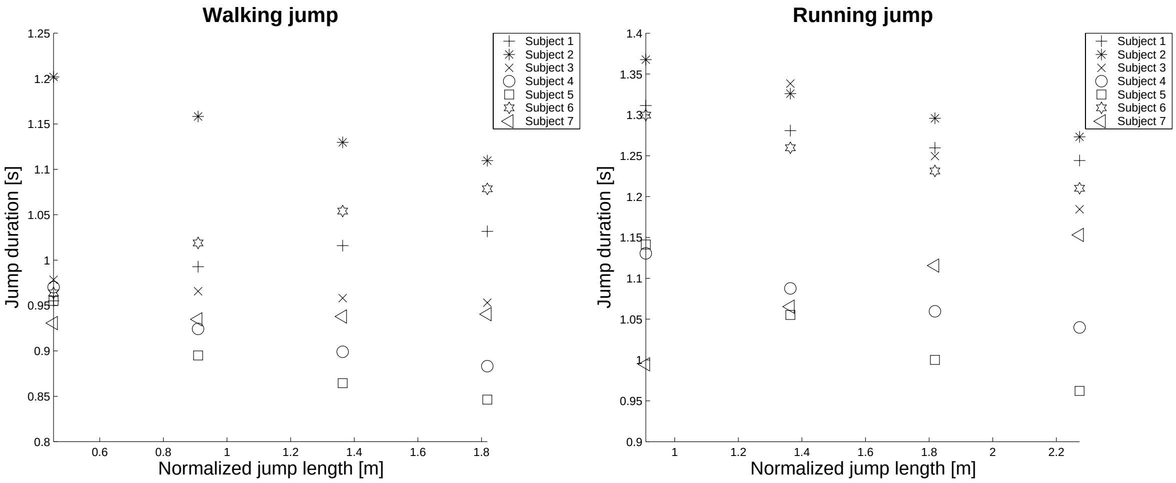


Figure 1.4: Computed jump duration according to parameterized synthesized jumps. **Left:** Walking jumps. **Right:** Running jumps.

Instead of computing this p_{root} on-the-fly when the frequency function has to be evaluated, we propose to approximate directly $F_{jump}^{walk,k}(l_n)$ with observed synthesized jumps. In our experiments, performed on various synthesized jumps, we compute their frequency with Eq. 1.9, using $f_{\bar{s}}^{run,k}(l_n)$ and Eq. 1.11. Fig. 1.4 shows a strong linearity, for a given subject k

and jump type, between the normalized jump length and their duration (inverse frequency). A linear least square is therefore computed between those values, leading to the following approximation (subject k, walking jump):

$$F_{jump}^{walk,k}(l_n) \approx a^{walk,k} l_n + b^{walk,k} \tag{1.12}$$

where $a^{walk,k}$ and $b^{walk,k}$ are the coefficient found by the least square fit. Analogously, the function $F_{jump}^{run,k}(l_n)$ is approximated for running jumps.

The general jump frequency function $F_{jump}(\mathbf{\Psi})$ is performed as follows. For a given w_{loco} value, which is either 0 or 1, the frequency is obtained by interpolating the $F_{jump}^{walk,k}(l_n)$ or $F_{jump}^{run,k}(l_n)$, weighted by the personification vector $\mathbf{w}_{subj}$. The final jump frequency is defined as:

$$F_{jump}(\mathbf{\Psi}) = \begin{cases} \sum_{k=1}^{N_{subj}} w_{subj,k} F_{jump}^{walk,k}(l_n) & \text{for } w_{loco} = 0 \\ \sum_{k=1}^{N_{subj}} w_{subj,k} F_{jump}^{run,k}(l_n) & \text{for } w_{loco} = 1 \end{cases} \tag{1.13}$$

To conclude, the presented locomotion and jump frequency functions have to verify that the update of the motion phase φ described in Eq. 1.2 is monotonic. This condition allows to avoid going back to the past, especially when strong speed variations occur. The phase increment $\Delta\varphi$ has therefore to be equal or greater than zero. This is verified as $\Delta t \geq 0$, and as $F_{loco} \geqslant 0$ and $F_{jump} \geqslant 0$ are ensured.

1.4 Continuous Motion Generation

In this section, we explain the procedure to generate a continuous animation from the generated normalized motion unit $\boldsymbol{\theta}(\mathbf{\Psi})$ by our motion modeling method.

In a pre-processing stage, the hierarchical PCA structure and the approximation functions in the lowest PCA spaces are computed, as described in Part II, chapter 2.2. For the locomotion, the structure is composed of one main PCA, N_{subj} sub-PCA level 1 (one per subject), and $2N_{subj}$ (walking and running) sub-PCA level 2 spaces. For the jump activity, two hierarchical structures are needed, one for each jump type. Those consist of one main PCA and N_{subj} sub-PCA level 1 (one per subject).

During the animation, we represent a motion as a continuous function over time

$$\boldsymbol{M}(t) = (\mathbf{p}_r(t), \mathbf{q}_r(t), \mathbf{q}_1(t), \ldots, \mathbf{q}_n(t)) \tag{1.14}$$

where $\mathbf{p}_r(t)$ and $\mathbf{q}_r(t)$ represent the global position and orientation of the root node and $\mathbf{q}_i(t)$, for $i > 0$, the local transformation of the i-th joint.

To compute $\boldsymbol{M}(t)$ at a given time $t = t_i$ (i.e. a posture), three stages are necessary: first, if $\mathbf{\Psi}$ has changed, a motion unit $\boldsymbol{\theta}(\mathbf{\Psi})$ has to be generated. Then the current time is wrapped into the normalized time to find the corresponding frame $\mathcal{F}_i$ in $\boldsymbol{\theta}(\mathbf{\Psi})$. Finally, the global translation and rotation are updated.

The first stage works as follows. At any time when the animator requires a motion parameter change, a new $\boldsymbol{\theta}(\boldsymbol{\Psi})$ is generated. As the methods described in our motion modeling do not consider the normalized parameter speed and jump length , the given speed s (respectively jump length l) from $\boldsymbol{\Psi}$ is substituted by s' (l') as follows:

$$s' = \frac{s}{H} H_{mocap} \quad l' = \frac{l}{H} H_{mocap} \tag{1.15}$$

where H corresponds to the leg length of the current animated character, and H_{mocap} of the character who had performed the original sequences.

A new motion unit is then generated according to this ”normalized” parameter vector $\boldsymbol{\Psi}$, by traversing the PCA structure form the lowest to the highest level. At each level, the coefficient vectors of the corresponding PCA spaces are computed with respect to one high-level parameter from $\boldsymbol{\Psi}$ and are used to go up in the hierarchy. This process is well described in Part II, Eq. 2.18 for the locomotion and Eq. 2.22 for jumps. To finalize this space normalization stage, all global translations $\mathbf{p}_{ri}$ from the frames $\mathcal{F}_i$ composing the new generated $\boldsymbol{\theta}(\boldsymbol{\Psi})$ are scaled by the leg length H. In fact, this is necessary as the original data have been normalized by the leg length of the captured subject.

The second stage consists in dynamically unwrapping the normalized time into the current time t_i, by updating the locomotion phase from the previous φ_{i-1} to the current φ_i. Actually, to the elapsed time $\Delta t = t_i - t_{i-1}$ corresponds a $\Delta\varphi$, computed as described in Eq. 1.2. The resulting phase φ_i is multiplied by the total frame number N_{frame} of a motion unit so as to obtain a frame index $idx = \varphi_i N_{frame}$. This index has to correspond to a specific frame $\mathcal{F}(\varphi_i)$ from $\boldsymbol{\theta}$. However, as the frames of $\boldsymbol{\theta}$ are regularly sampled, we define $\mathcal{F}(\varphi_i)$ as:

$$\mathcal{F}(\varphi_i) = \text{SLERP}(\mathcal{F}_{\lfloor idx \rfloor}, \mathcal{F}_{\lceil idx \rceil}, \frac{idx - \lfloor idx \rfloor}{\lceil idx \rceil - \lfloor idx \rfloor}) \tag{1.16}$$

where SLERP(a, b, p) defines the spherical linear interpolation [Shoemake, 1985] between the joints of the posture a and p, with an interpolation weight p. Hence, we can define a continuous function $\boldsymbol{\theta}(\boldsymbol{\Psi}, \varphi_i)$ of the motion unit $\boldsymbol{\theta}(\boldsymbol{\Psi})$ as:

$$\boldsymbol{\theta}(\boldsymbol{\Psi}, \varphi_i) = \mathcal{F}(\varphi_i) = (\widehat{\mathbf{p}}_r(\varphi_i), \widehat{\mathbf{q}}_r(\varphi_i), \mathbf{q}_1(\varphi_i), \ldots, \mathbf{q}_n(\varphi_i)) \tag{1.17}$$

We remind that $\widehat{\mathbf{p}}_r(\varphi_i)$ and $\widehat{\mathbf{q}}_r(\varphi_i)$ contain only the translation and orientation of treadmill locomotion.

Finally $\boldsymbol{M}(t_i)$ has to be constructed in the global coordinate system as the root node of $\boldsymbol{\theta}$ is expressed in a local coordinate system aligned with the body. In addition, this node contains in case of locomotion only local translation and orientation oscillations of the motion, as the original cycles of the motion model are performed on a treadmill. Therefore, according to the linear velocity $\mathbf{v}$ (whose norm $||\mathbf{v}||$ equals s), $\mathbf{p}_r(t_i)$ and $\mathbf{q}_r(t_i)$ of $\boldsymbol{M}(t_i)$ are modified with respect to the elapsed time $\Delta t = t_i - t_{i-1}$:

$$\begin{aligned} \mathbf{p}_r(t_i) &= \mathbf{p}_r(t_{i-1}) - \widehat{\mathbf{p}}_r(\varphi_{i-1}) + \widehat{\mathbf{p}}_r(\varphi_i) + \mathbf{v}\Delta t \\ \mathbf{q}_r(t_i) &= \mathbf{q}_r(t_{i-1}) * \widehat{\mathbf{q}}_r(\varphi_{i-1})^{-1} * \widehat{\mathbf{q}}_r(\varphi_i) \end{aligned} \tag{1.18}$$

For the jump motions, the speed value is neglected as the motion's global translations have been effectively captured.

1.5 Results

Concretely, on-the-fly high-level parameter can be changed, producing a smooth update of the motion in real-time. When the human size is scaled, the cycle frequency is directly influenced in order to avoid feet sliding. Fig 1.5 illustrates a walking cycle with a continuous human size variation.

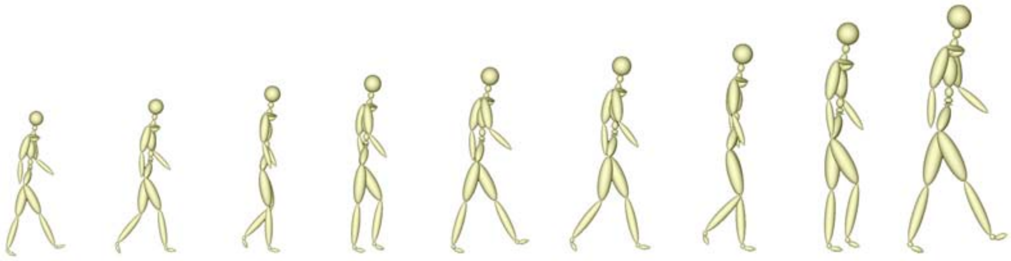


Figure 1.5: A walking cycle with a continuous human size variation.

By varying the speed s, the presented motion model generates locomotion from 0 up to 12 [km/h] for walking patterns (see Fig. 1.6), and from 0 to 17 [km/h] (see Fig. 1.7), wide beyond the original data.

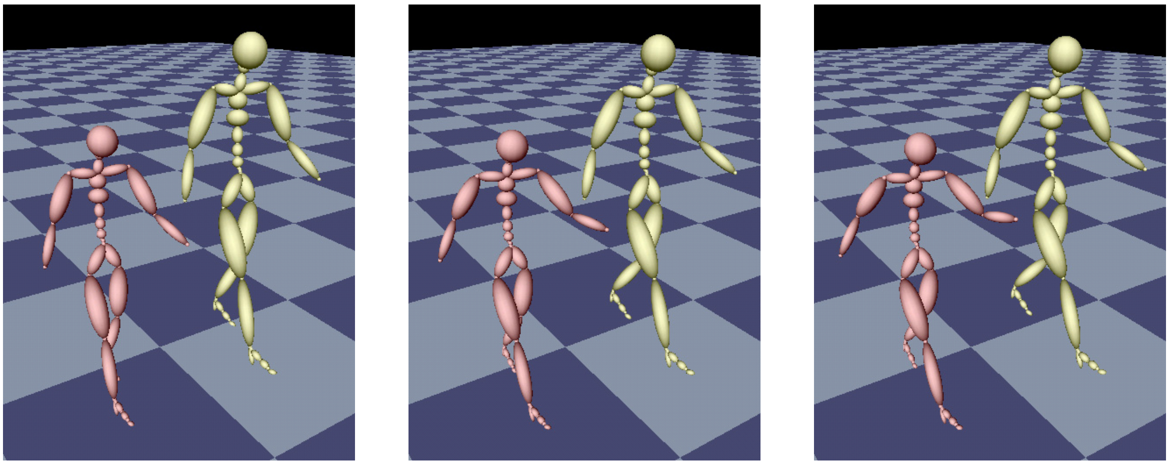


Figure 1.6: Postures of walking at identical instant, with two virtual humans having different size and personification parameters. From left to right, only the speed varies: 2.0, 6.0 and 10.0 [km/h].

Outside those bounds, undesired behaviors appear. First, the lower arms reach the head level with the elbows having an unrealistic rotation. Secondly, the double support phase for walking is no longer ensured. The transition from walking to running (and inversely) obtained by varying continuously the parameter w_{loco} is seamless, modifying the dynamics of the movements and the arm trajectories, which are higher for running.

Finally, the personification parameter $\mathbf{w}_{subj}$ is less easy to exploit, as we did not capture exaggerated locomotion patterns such as lazy, happy or explosive. Techniques presented

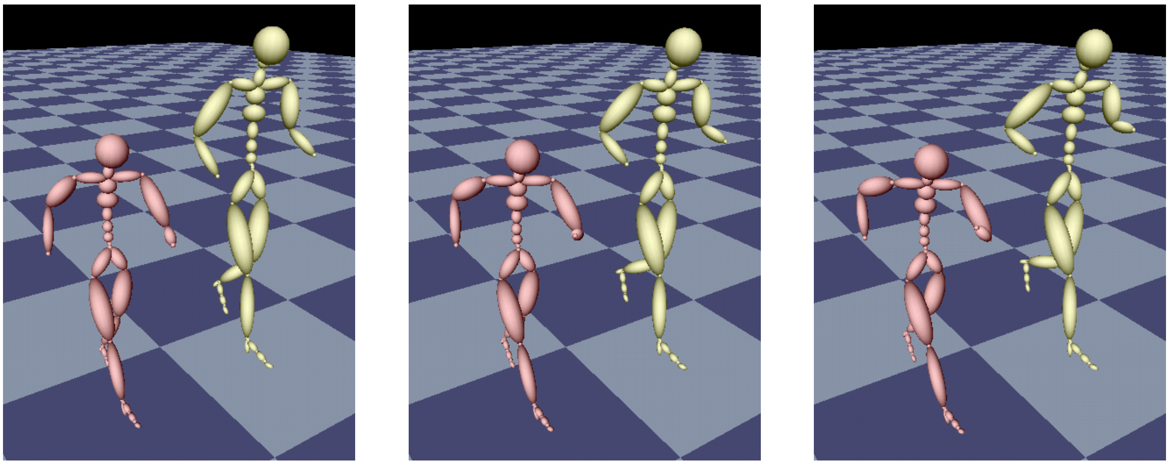


Figure 1.7: Postures of running at identical instant, with two virtual humans having different size and personification parameter. From left to right, only the speed varies: 4.0, 8.0 and 18.0 [km/h].

in [Neff and Fiume, 2003, 2005] can be applied to make the animated characters more expressively, by modifying the motion amplitude for example. Nevertheless, the weights associated to the subjects can be mixed to obtained new stylized motions. These are preserved over the speed range and locomotion weight because the hierarchical PCA structure decouples speed, locomotion weight and personification. Fig. 1.8 illustrate two personification weight combinations, inducing essentially different arm trajectories.

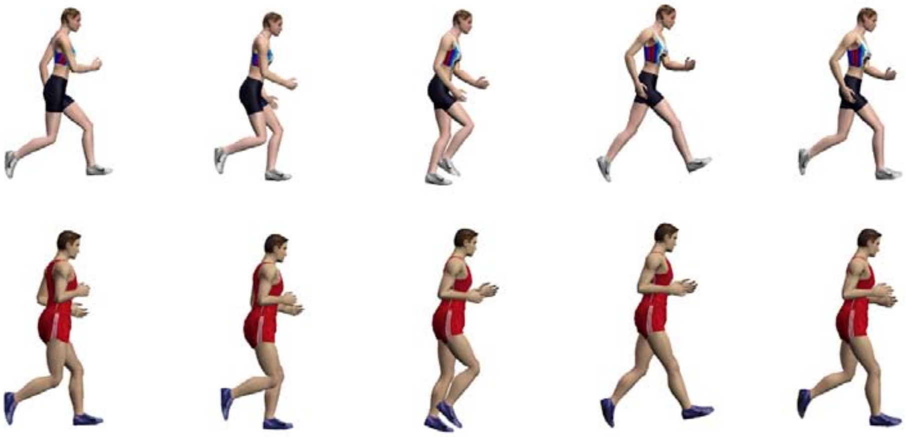


Figure 1.8: The sequence of a half running cycle for two different personification vectors.

The generation of jump sequences is different, as the parameter vector $\mathbf{\Psi}$ is not modified while the jump is performed. The jump length can be varied from 0.3 to 1.7 [m] for walking jump (see Fig. 1.9). From the original captured data, a jump can be extrapolated at maximum with a length of 40% bigger than the maximal captured jump length. For running jumps, the jump length vary from 0.5 until 2.0 [m] (see Fig. 1.10), representing a maximal jump length of 25% bigger than the maximum captured jump length. In practice, the longer the jump, the more the chest tilts forward and the step length increases. The human size can be changed.

Thanks to the normalization of the jump length according to the leg length, for a given jump length, a smaller human has to perform a more ”difficult” jump than a bigger human. The personification parameter $\mathbf{w}_{subj}$ is more relevant to use as in the locomotion case. The database contains more diversity, and therefore allows to distinguish different jump styles.

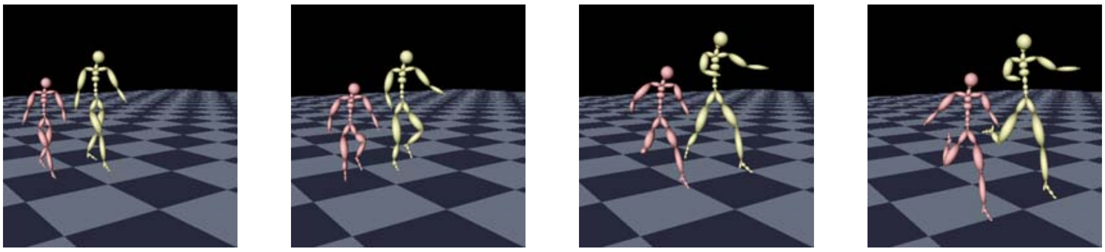


Figure 1.9: Sequence of a walking jump of two virtual humans having different size and personification parameters. The jump length is 1.4 [m], beyond captured data.

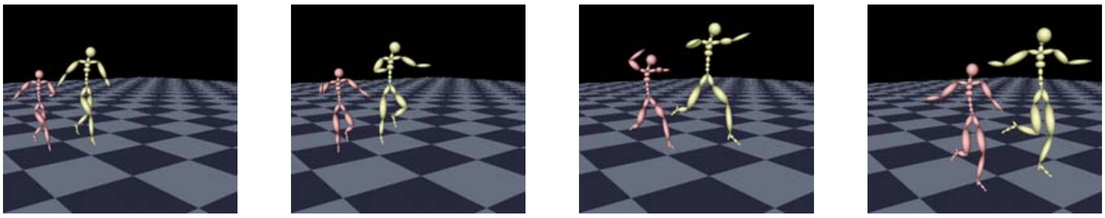


Figure 1.10: Sequence of a running jump of two virtual humans having different size and personification parameters. The jump length is 1.7 [m], beyond captured data.

In terms of performance, 24 sec are necessary to construct the PCA structure for locomotion and jumps, and to compute the approximation functions. The updating for the generation of a new motion is performed within 0.23 milliseconds on a CPU 1.8 GHz machine. As our model computes the entire locomotion cycle, we can infer that one frame can be potentially updated in less than 0.01 milliseconds.

1.5.1 Validation

The validation of the resulting generated animation is firstly achieved by visualizing them. They look smooth and realistic. However, even if the human eye is a good judge, we proposed an objective approach, by comparing the difference between original and synthesized motions. Motion comparison techniques propose two approaches. The first one, presented in [Kovar et al., 2002a] measures the distance between two motions by considering 3D points placed on the body. Another metric is described in [Lee et al., 2002] which measure difference of joint angles and velocities. For these two techniques, weights can be assigned to the body parts, in order to give more influence to specific body's parts. However, it remains difficult to find an appropriate weight configuration [Wang and Bodenheimer, 2003].

We choose a metric based on the joint angles difference referred to as the geodesic norm [Buss and Fillmore, 2001]. For each frame to compare, we square the Euclidean norm of the difference between original and generated joint rotations, defined with exponential maps (see Part II, Eq. 2.1). The metric function $d_{rot}(\mathbf{q}_1, \mathbf{q}_2)$ which measures the difference between two rotations expressed by the unit quaternions $\mathbf{q}_1$ and $\mathbf{q}_2$ is therefore computed as:

$$d_{rot}(\mathbf{q}_1, \mathbf{q}_2) = \left|\left|\ln(\mathbf{q}_1\mathbf{q}_2^{-1})\right|\right|^2 \tag{1.19}$$

For two motions to be compared, this metric function is applied to every joint of the body. The results can be represented in a distance matrix where each cell contains the computed difference for a given joint, at a given frame. Owing the difficulty to set weights on the body joints, we modify the distance matrix by selecting the most relevant joints for our animation context: shoulder and elbow joints for the arms; hip, knee and ankle joints for the legs. In order to be able to quantify this metric, we compare two very different motions in Fig. 1.11, namely a slow walking with a fast running.

Fig. 1.12 illustrates two comparisons between the mean of four original cycles and the corresponding synthesized one: walking at 5.5 [km/h] with subject $k = 3$ and running at 8.0 [km/h] with subject $k = 2$.

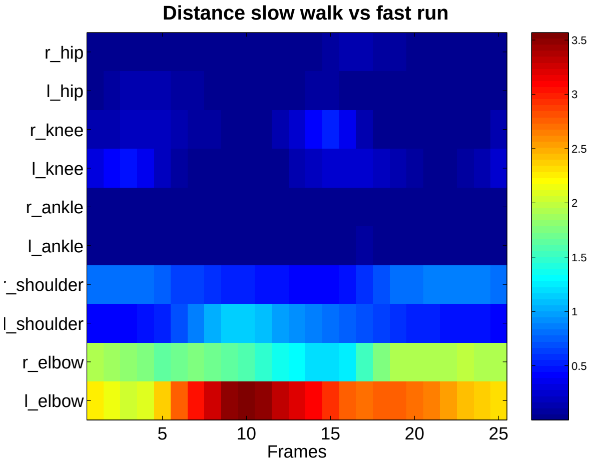


Figure 1.11: Distance between original slow walking (3.0 [km/h]) and a fast running motion 11.0 [km/h]) for a same subject.

To investigate the complete database, our metric function should be applied on all original motions, compared with their corresponding synthesized version. However, to get a more compact overview of the results, we propose a new metric function d_{motion} which returns the distance of two compared motions. For a given pair of captured (or original) $\boldsymbol{\theta}_c(\boldsymbol{\Psi})$ and generated (or synthesized) $\boldsymbol{\theta}_g(\boldsymbol{\Psi})$ motion, the metric function $d_{motion}\left(\boldsymbol{\theta}_c\left(\boldsymbol{\Psi}\right), \boldsymbol{\theta}_g\left(\boldsymbol{\Psi}\right)\right)$ is defined as follows:

$$d_{motion}(\boldsymbol{\theta}_c, \boldsymbol{\theta}_g) = \frac{1}{N_{joints}} \sum_{i=1}^{N} \left[\frac{1}{f_{max}} \sum_{j=1}^{N_{frame}} d_{rot}(\mathbf{q}_{c_{ij}}, \mathbf{q}_{g_{ij}}) \right] \tag{1.20}$$

where $\mathbf{q}_{c_{ij}}$ and $\mathbf{q}_{s_{ij}}$ are the i-th rotations at the j-th frame, N_{joints} the measured joint number.

This process is performed on all walking cycles, and their d_{motion} are stored in a matrix. Each column determines a walking speed, and each lines an original motion, classified by

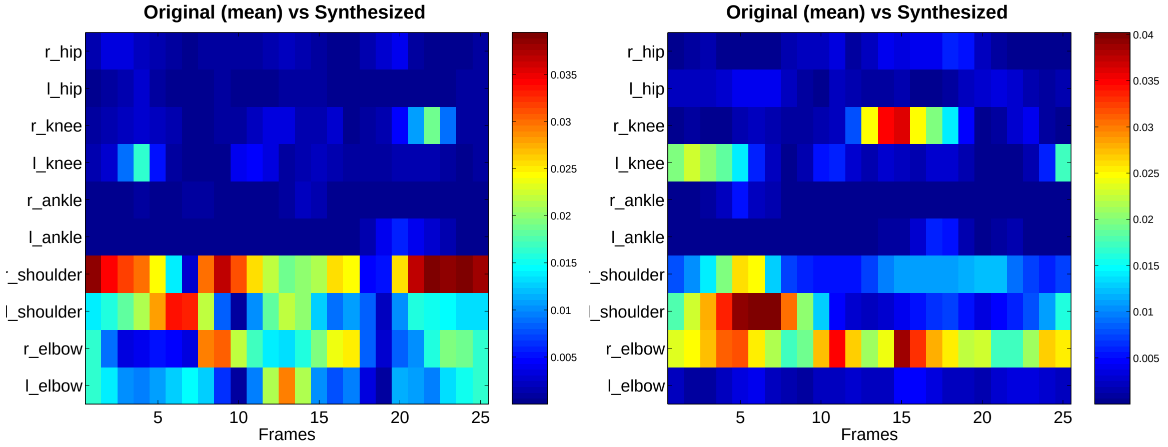


Figure 1.12: Distance between the mean of four original cycles and the corresponding synthesized one. **Left:** subject $k = 3$, walking at 5.5 [km/h]. **Right:** subject $k = 2$, running at 8.0 [km/h].

subject (Fig. 1.13, left). To quantify the numeric results of this matrix, a second one is constructed, representing the maximum variance of the original captured data having the same parameters. Each cell of this matrix represents the maximum distance d_{motion} between the original cycles of a same subject at a given speed value (Fig. 1.13, right). We observe that the generated motions with our method are very similar to the original ones, demonstrating the quality of results. Actually, the variation between an original and a synthesized cycle is approximately 30% smaller than the maximal variation between the corresponding original cycles. For walking motions at high speed (above 6 km/h), the variation becomes bigger. This is probably due to the difficulty for the performer to keep a constant style at such unnatural walking speed.

The same matrices are constructed for the running motions. Conclusions similar to the walking case can be drawn: the variation between an original and a synthesized cycle is approximately 20% smaller than the maximal variation between the four corresponding original cycles. In addition, the variation between the running original cycles is roughly identical to the one in case of walking, except the subject $k = 4$ at 12 km/h which holds the biggest difference.

We carry out the same distance comparison for the jump motions. Original motions are classified by their length, in three groups corresponding to the asked jump length during the motion capture sessions: length near to 0.6, 1.0 and 1.4 [m] for the walking run-up, and near to 1, 1.4 and 1.8 [m]. To each of these original motions we compare a synthesized jump having the same length. The computed distance is stored in a matrix, whose results for the walking jumps and running jumps are depicted in Fig. 1.15 respectively Fig. 1.16. In absolute, we observe bigger variance for the walking jumps than for running jumps. These differences may be explained by the difficulty to reproduce several jumps of the same length with an identical style, especially in the walking jump as the initial speed is rather slow.

Finally, the length of synthesized jumps is measured and compared to the one given as motion parameter in the model, for each subject. The results are illustrated in Fig. 1.17. Only two subjects ($k = 2$ and $k = 3$) show a curve which is not parallel to the ”ideal case”,

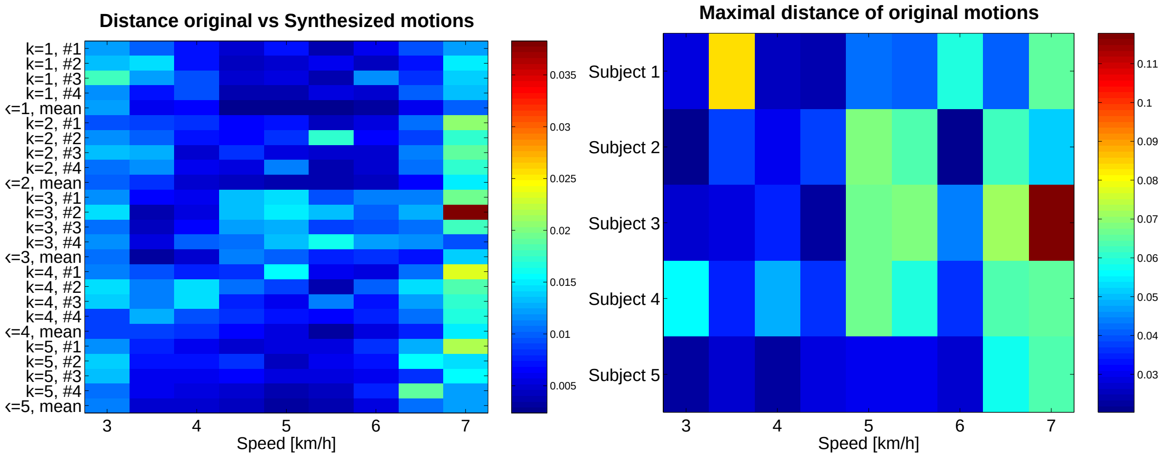


Figure 1.13: Validation of the synthesized walking motions: the distances between the synthesized motions and the corresponding original ones are smaller than the maximum distance of the four original (note that the different color scaling for both graphs). **Left:** The distance d_{motion}, at given speeds, between the synthesized and its corresponding four original motions (labeled from #1 to #4). **Right:** The maximum d_{motion}, at given speeds and subject k, between the corresponding four original motion samples.

representing a perfect matching between required and measured jump lengths. However, we observe that at least in the captured domain the results are good. In the running case, again two subjects ($k = 3$ and $k = 7$) produce less satisfying results. Beyond the captured domain, we notice excellent outcomes for big length. For small lengths, the jumps are not approximated that well, as it is not natural to perform such jumps in real life.

1.5.2 Discussion

In this section, we discuss first the scalability of our method, and secondly the quality of the hierarchical PCA structure by concretely comparing it to another approach, using the proposed metrics.

The dimension of our motion database has to be large in order to apply PCA. In fact, this method is based on statistics and needs enough sample data to validate it. However, the sample number in each of the lowest PCA spaces can be at least decreased to two samples so as to compute their associated linear approximation functions. We have tried our model with fewer motions as in our original database, producing similar results. Nevertheless, possible outliers in the lowest PCA spaces have to be identified as the approximation method will be more sensitive to them, especially with very few sample number.

On the contrary, the dimension database can be increased by adding new captured subjects for example. This is interesting to produce more stylized motions. As a consequence, the hierarchical structure has to be recomputed and the number of retained PCs from the main PCA space becomes bigger. Its progression is dependent on the inter-variability between the captured subjects, as more PCs are necessary to explain significant motion diversity. We have tested the progression of the number of PCs by adding incrementally the motions of

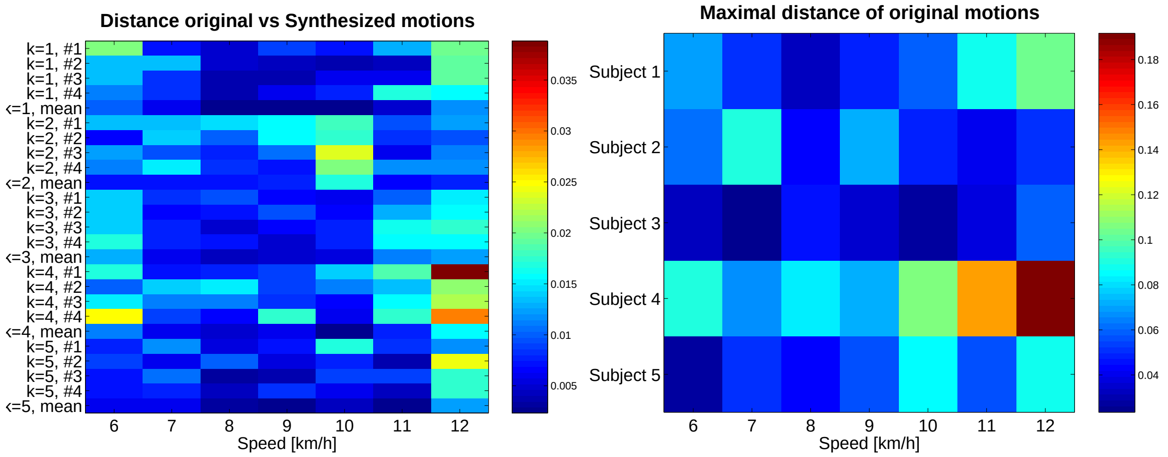


Figure 1.14: Validation of the synthesized running motions: the distances between the synthesized motions and the corresponding original ones are smaller than the maximum distance of the four original (note that the different color scaling for both graphs). **Left:** The distance d_{motion}, at given speeds, between the synthesized and its corresponding four original motions (labeled from #1 to #4). **Right:** The maximum d_{motion}, at given speeds and subject k, between the corresponding four original motions.

one new subject in our motion matrix. The results are summarized in Table 1.2, indicating a logarithmic tendency.

# subject	# PCs
1	3
2	5
3	7
4	9
5	9

Table 1.2: The number of retained PCs (to represent 90% of the original data) with respect to the number of subject in the motion matrix.

Another approach presented in [Urtasun et al., 2004] allows the addition of a new performer without re-computing the hierarchical structure. This approach improves the personification diversity, by using a single sample of a new subject. This sample is projected onto our main PCA space. The authors use our motion modeling to generate all possible motions from all existing N_{subj} subjects having the same parameters as the projected one. A metric based on the Mahalanobis distance is used to measure the distance between those motions and the projected one. The normalized distance between the i-th subject corresponds to the weight $w_{subj,i}$ of the personification vector $\mathbf{w}_{subj}$. Hence, $\mathbf{w}_{subj}$ is fully determined to produce the projected motion. The other parameters w_{loco} and s (or l) can be varied in order to generate new motions with respect to the new subject.

The second point of this discussion focuses on the hierarchical PCA structure, presented in Part II, Chapter 2. Instead of it, one could classify the motion by subject and type of

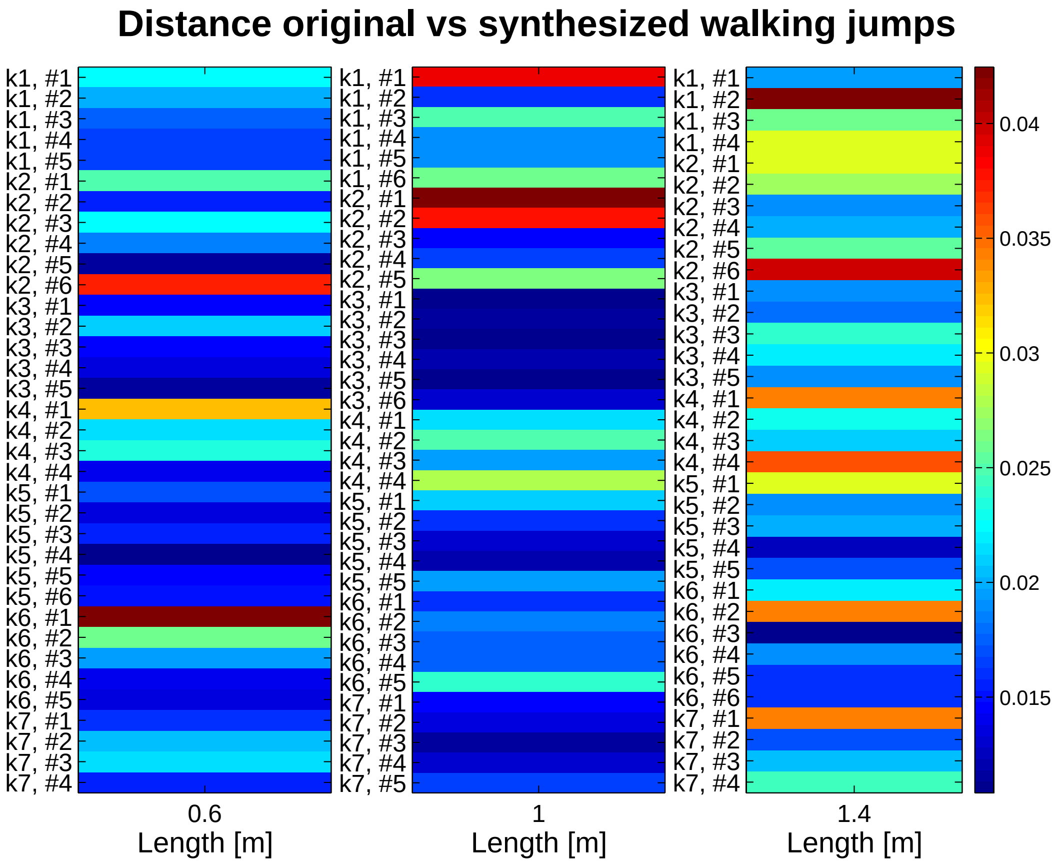


Figure 1.15: Validation of walking jump motions: comparison between original and synthesized motions, classified by k-th subject and length.

locomotion directly at the main PCA level and compute the approximation functions in this space. This alternative, referred as ”flat” structure, has three points which work against it.

First, the hierarchical approach is the most intuitive way to match high-level parameters with PCs. By applying this structure, the correspondence can be visualized in a very small dimension (one or two). It allows therefore to get a better control to judge easily the quality of the linear approximation.

Secondly in terms of computational cost, the flat structure method is more expensive. In fact, the PCs number of the main PCA is strongly related to the subject number. This directly affects the computational cost of the linear least square fit, achieved in the dimensions of the main PCA space. On the contrary, we can assume that for the hierarchical method, independently of the subject number, the sub-PCA level 1 has at most two PCs and only one in the sub-PCA level 2. This hypothesis is confirmed by our experiments and the fact that motions of a same subject do not contain high variance. Actually, depending on the main PCA dimension, the number of operations to compute a new motion in the flat structure is at most twice as much as the hierarchical structure.

Finally, we compare the results produced by both methods, for similar parameterized original motions. The metric presented in Eq. 1.20 is used to compare between the synthesized motion and the mean of original motions for a given parameter vector $\mathbf{\Psi}$. In the

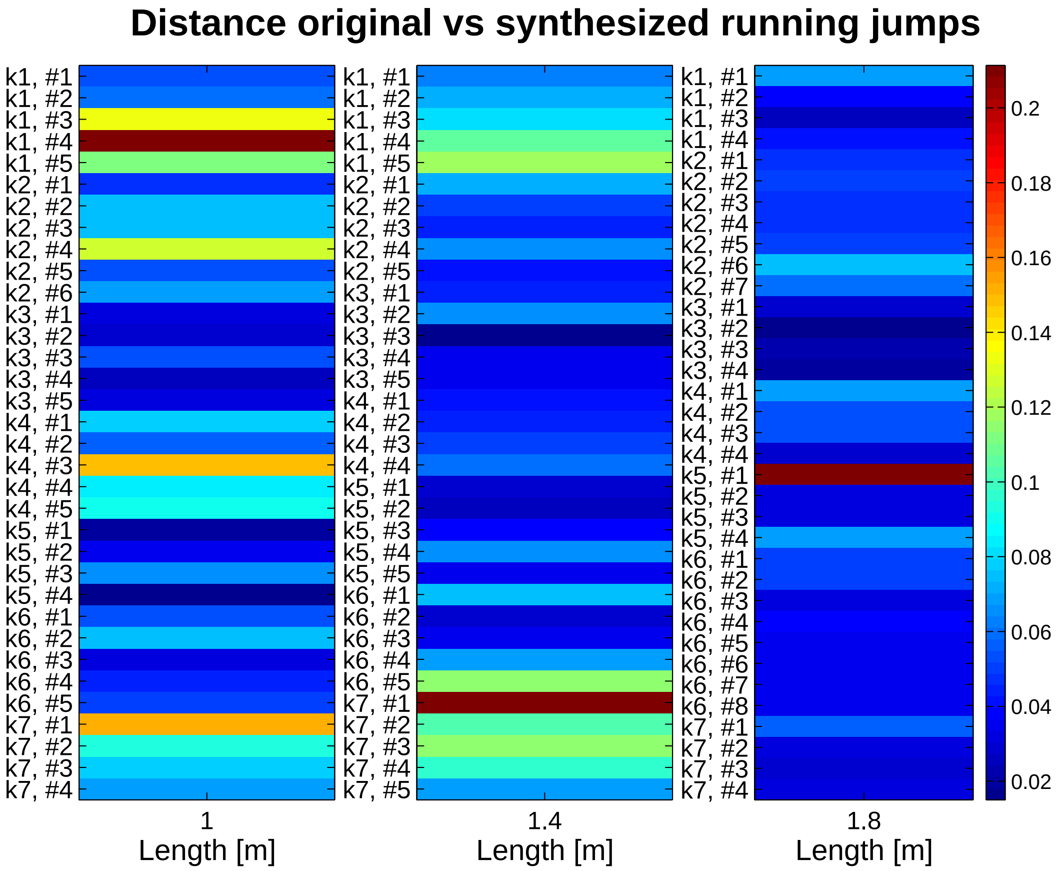


Figure 1.16: Validation of running jump motions: comparison between original and synthesized motions, classified by k-th subject and length.

convex hull defined by the captured motions, we observe that the flat method emphasizes the differences observed when applying the hierarchical structure (see Fig. 1.18 for an example). Therefore, the hierarchical structure method produces more reliable motions. For motions generated beyond the range of captured data, the major difference concerns the upper body parts, accentuated in case of running motions, as illustrated in Figure 1.19. The lower body joints contain fewer differences, but could produce different step length and therefore foot sliding. In practice, these differences are difficult to notice.

1.6 Conclusion

In this chapter, we have improved the motion modeling method so as to produce character animation with continuous high-level parameter variation. Thanks to a normalized space, our method is adaptable to any virtual human size. In addition, the time normalization method retrieves the original animation duration for locomotion cycles as well as jump sequences with respect to the user parameters.

To our knowledge, no work has proposed a real-time parameterized jumping engine

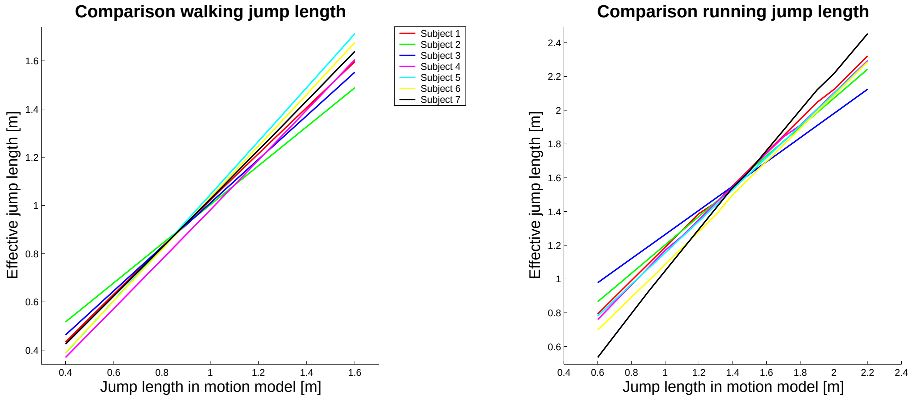


Figure 1.17: Comparison between the given jump length l introduced as parameter and the effective synthesized jumping length. **Left:** Walking jumps. **Right:** Running jumps.

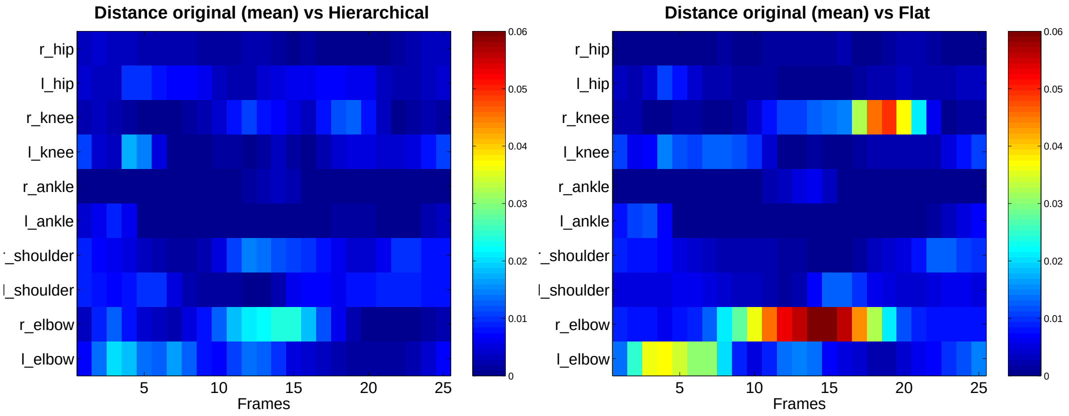


Figure 1.18: Comparison of the distances between the mean of four original walking motions and their corresponding synthesized motion. **Left:** The synthesized motion is generated using the hierarchical method. **Right:** The synthesized motion is generated using the flat method.

based on motion capture data. Works such as [Safonova et al., 2004; Liu and Popović, 2002] include physics but motions are not computed in real-time. Nevertheless techniques based on scattered data interpolation [Rose et al., 1996] or blending [Park et al., 2002a] may be applied. In this case, their associated time-warping methods require that the user specify the duration of a generated jump as a supplementary parameter, in order to distinguish it from others having a same length. Another technique presented in [Kovar and Gleicher, 2003] can be applied to jumping motions. The animator would have to specify implicitly the jump duration, by assigning different weight values to original jumps of similar length.

The resulting animations demonstrated the ability of the method to produce new motions inside the convex hull of the motion capture database, as well as outside. The effective jump lengths of the synthesized sequences have been measured and compared to the given user parameters, leading to excellent results except for two subjects in case of running jumps. The resulting animations have been also validated by comparing them with the original captured ones. We proposed a distance function based on joint rotation difference to quantify the variance between the synthesized and original motions. The results are good, as this vari-

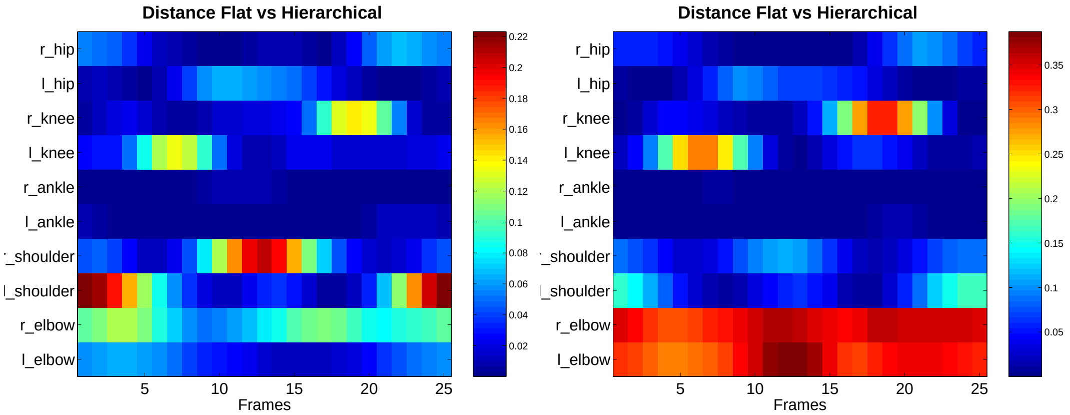


Figure 1.19: Distances between the hierarchical and flat method with motion parameters beyond captured data. **Left:** Walking motion at 10 [km/h]. **Right:** Running motions at 15 [km/h].

ance for a given parameter vector is approximately twice smaller than the maximum variance between the original motions with similar parameters. Further investigations can be applied to our synthesized motions, by evaluating their visual fidelity. Quality metrics or heuristics are needed to achieve this, but the quality is often difficult to define. In [O'Sullivan et al., 2003], metrics are proposed to evaluate simple animations of objects, while recently Ren et al. [2003] present metrics which evaluate human animations from motion capture. In addition, Safonova and Hodgins [2005] analyze the physical correctness of motion interpolation and suggest small modification for more natural looking animations.

Our metric function is also applied to balance the hierarchical PCA structure against another approach, a flat structure based on a single PCA level. In addition to save computational cost, the hierarchical structure has shown less variance compared with the flat structure.

One of the interesting aspects of the presented method is that motions are not considered as a sequence of frames, but as a complete entity. Due to the dimension compression, the performance to update the current motion is similar, and even better than traditional per-frame continuous animation methods [Rose et al., 1996; Park et al., 2002a; Kovar and Gleicher, 2003]. Our approach is particularly appropriate to achieve motion anticipation in order to control the foot constraints for example, as presented in the next chapter.

Chapter 2

On-line Adaptive Footplant Detection and Enforcement

2.1 Introduction

It results from the previous chapter that our motion engine method generates in real-time parameterized motions, fulfilling therefore a fundamental property in character animation. Actually, it is necessary, on the one hand, to produce animations controlled by high-level parameters, like changing the locomotion style and speed to walk around a virtual environment. On the other hand, small continuous variations of those parameters increase and sustain the believability of character movements. These variations have to be performed on-line, reactive to user's requests or to autonomous agents.

Concurrently, the resulting animation should be as realistic as possible, notably by maintaining basic physical constraints. Among them, keeping the foot planted on the floor during a period a time (referred to as *footplant* in this thesis) is a common problem in virtual character animation. This foot-floor constraint preservation has entailed the elaboration of numerous methods, divided into two distinct stages: the detection of a footplant and its enforcement.

The traditional methods [Bindiganavale and Badler, 1998; Menardais et al., 2004] which detect the start and end of a footplant, use global thresholds on the position and velocity of the feet, independently of the motion type containing the footplants. However, threshold parameters for a light walk are not compatible with a high dynamic run. In addition, these methods assume that the motions are free from noise and artifacts such as foot sliding. Therefore, in this chapter, we tackle a first problem: the footplant detection in real-time. We propose an adaptive on-line method which takes into account the nature and the quality of the motion to determine threshold values automatically. In addition, this detection technique utilizes the motion structure to determine restricted periods of time when a constraint has to be investigated.

Concerning the second stage, many methods enforcing footplants are based on specialized IK (Inverse Kinematics) solvers [Lee and Shin, 1999; Shin et al., 2001; Kovar et al., 2002b]. All, except for the method of Kovar et al. [Kovar et al., 2002b], can be applied on-line, for example for motion retargeting [Park et al., 2002a] or to maintain foot constraints during the transition between two motion capture clips [Lee et al., 2002]. However, these approaches are unadapted when the original position needs to be modified at the constraint time. In practice, imperfect input motions may contain a footplant whose foot trajectory ends above the ground. In this case, the constraint position has to be re-positioned on the ground, while ensuring a smooth motion correction. Hence, we address a second problem in this chapter: real-time constraint re-positioning and enforcement. We correct a posture with the new constraint positions by using a numerical IK algorithm, robust enough to work on-line. In order to avoid an abrupt change in the constraint location, we introduce an ease-in phase based on a displacement map, allowing to modify the motion towards the new posture smoothly. Similarly, an ease-out phase allows to seamlessly release the constraint. In addition, we improve the motion realism by defining two effectors (heel and toe) on each foot which control the footplant.

To deal with the two main problems presented in this chapter, we introduce an approach based on anticipation. As described in [Berthoz, 2000], the human's brain simulates and anticipates the future in order to adjust the present. This proactive concept is modeled in our method by obtaining future frames of the current animation in order to react according to the future. Actually, as the constraint detection method is adaptive (i.e. the method adapts its parameters according to the current foot trajectory), it needs to know the foot trajectory in advance. For footplant enforcement, the corrected posture at the start of the constraint is required beforehand. In this case, future motion information is also crucial. However, the motion is generated on-line, producing a continuous stream of frames with uncertain variations due to the parameter changes. Therefore, to compensate these two paradoxical goals, we introduce a novel approach capable of anticipating the motion. According to the motion state, described with its current and desired high-level parameters, our method computes future information as to react correctly in the present.

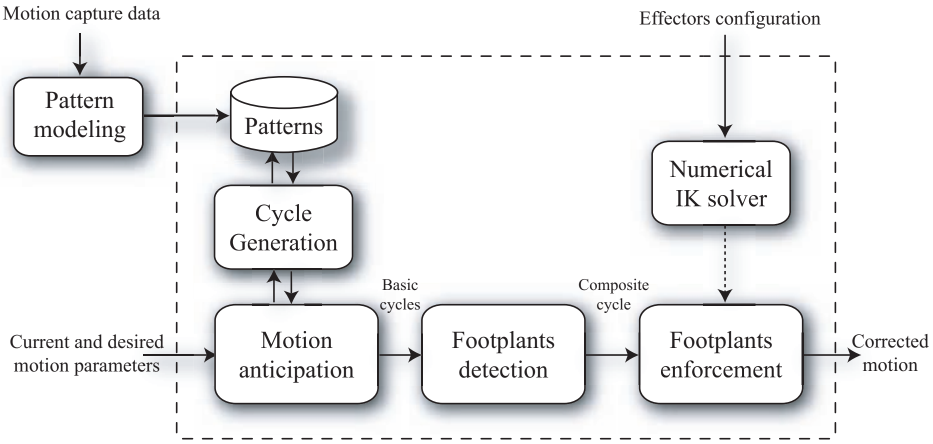


Figure 2.1: System overview

A complete overview of our system is depicted in Fig. 2.1. To illustrate our methodology, we focus on locomotion patterns, providing walking and running cycles. At any time, the user can determine either current high-level parameters, or desired ones which have to be reached after a specific period of time. In addition, effectors can be configured in order to modify the foot constraint location. According to the varying parameter set, the motion anticipation module computes multiple anticipated frames, each of them stemming from motion units according to the parameter variation. Footplants are also detected on those motion units, and are enforced using a numerical IK solver. In addition, we refer the postures from anticipated frames to as a composite cycle. Postures from this cycle are used to ensure a smooth transition from the unconstrained to the constrained state.

2.2 Motion Anticipation

We define the term of motion anticipation as the ability to generate future motion postures. In the case of off-line animation processes, these postures are directly at disposal. The particular case of on-line motion generation is much more complicated. In fact, traditional animation methods [Rose et al., 1998; Park et al., 2002a] compute the postures frame by frame, according to an elapsed Δt time between two animation updates. To anticipate postures, it is necessary to have a method which is efficient enough not to alter the real-time motion generation process, in terms of update rate. In addition, the anticipation has to take into account possible parameter variations, such as changes in the locomotion speed.

To address on-line anticipation, we base our approach on the motion modeling method presented in the previous chapter which generates a motion as a whole locomotion cycle as opposed to the standard frame by frame approach. Actually, the PCA algorithm considers *eigencycles* and not *eigenframes*. Therefore, an entire time-normalized walking or running cycle is computed at once, according to given motion parameters.

2.2.1 Current Posture Computation

The posture of a character at any time t_i of an animation is computed using $\boldsymbol{M}(t_i)$, as Eq. 1.14 describes in Part III, Section 1.4. We introduce a new parameter, the angular speed ω, in order to generate curved motions. From Eq. 1.18, the global position $\mathbf{p}_r(t_i)$ and orientation $\mathbf{q}_r(t_i)$ of $\boldsymbol{M}(t_i)$ are therefore modified with respect to the elapsed time $\Delta t = t_i - t_{i-1}$:

$$\begin{aligned} \mathbf{p}_r(t_i) &= \mathbf{p}_r(t_{i-1}) - \widehat{\mathbf{p}}_r(\varphi_{i-1}) + \widehat{\mathbf{p}}_r(\varphi_i) + \mathbf{v}\Delta t \\ \mathbf{q}_r(t_i) &= \mathbf{q}_r(t_{i-1}) * \widehat{\mathbf{q}}_r(\varphi_{i-1})^{-1} * \widehat{\mathbf{q}}_r(\varphi_i) * Rot(\omega\Delta t) \end{aligned} \tag{2.1}$$

where $Rot(\alpha)$ defines the rotation of the yaw angle α.

2.2.2 Anticipated Posture Computation

To anticipate the motion, our method generates not only $\boldsymbol{M}(t_i)$, but also any future posture $\boldsymbol{M}(t_i + \Delta T)$, in real-time and at any point in time t_i. Fig. 2.2 illustrate the anticipation of 13 postures, with a constant time interval between each posture. The yellow bodies at the leftmost on the images represent the current postures, whereas the others are anticipated. To explain the posture computation, we consider two contexts: one where the parameter vector $\boldsymbol{\Psi}$ does not vary (Fig. 2.2, left), and one where it continuously varies (Fig. 2.2, right).

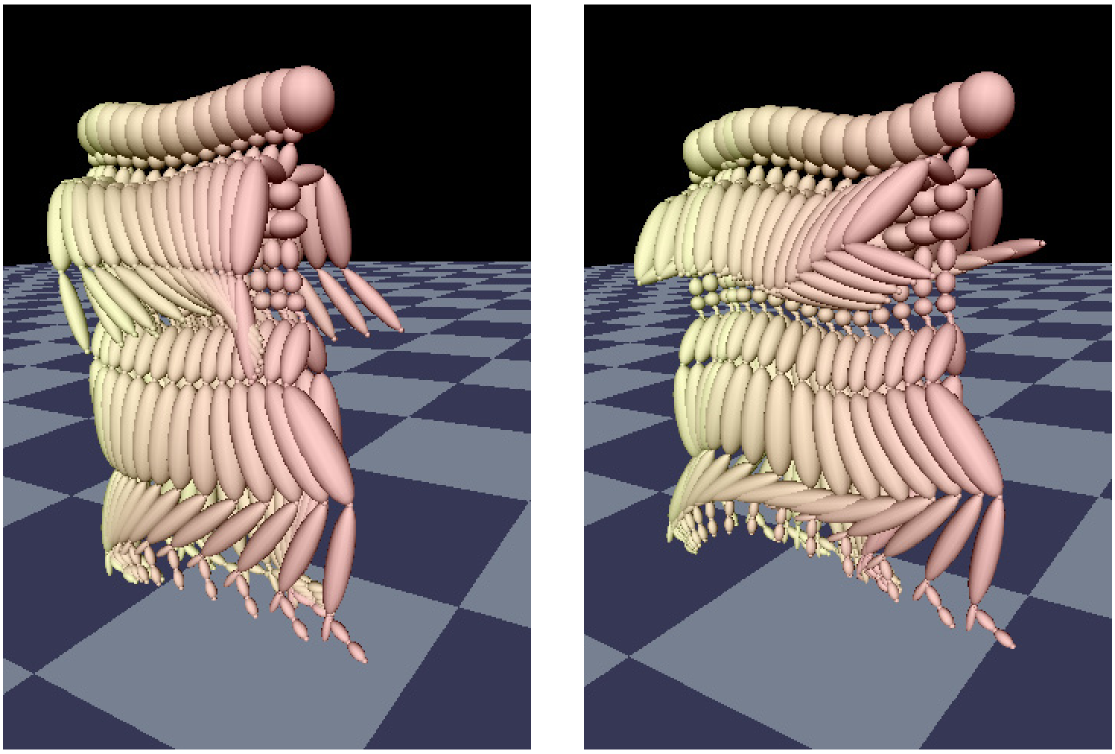


Figure 2.2: Motion patterns with anticipation. The yellow posture (on the leftmost) represents the current frame, while the others (from yellow to red, towards right) are anticipated postures. **Left:** Invariant motion parameters.**Right:** Variation of the speed parameter.

We propose the creation of a buffer containing n anticipated postures, with a constant time interval between postures. For the first context (we assume that the angular speed is null), the j-th posture $\boldsymbol{M}(t_i + j\Delta T)$, for $j = 1 \ldots n$, is computed analogously to $\boldsymbol{M}(t_i)$. As $\boldsymbol{\theta}(\boldsymbol{\Psi})$ contains all the cycle frames, only the root node has to be updated according to Eq. 2.1, by substituting ΔT for Δt.

The second context considers a continuous variation of $\boldsymbol{\Psi}$. Let $\boldsymbol{\Psi}_j$ be the parameter vector at time $t_j = t_i + j\Delta T$. Hence, the computation of the j-th posture at time t_j involves the generation of a new motion pattern $\boldsymbol{\theta}(\boldsymbol{\Psi}_j)$. We refer to the set of all these n new patterns as composite cycles. Then, the root node of each $\boldsymbol{M}(t_j)$ is sequentially updated, starting from $j = 1$ until $j = n$, to take into account for the continuous parameter variation. Eq. 2.2 described one update step, where $\Delta T = t_j - t_{j-1}$.

$$\begin{aligned} \mathbf{p}_r(t_j) &= \mathbf{p}_r(t_{j-1}) - \widehat{\mathbf{p}}_r(\varphi_{j-1}) + \widehat{\mathbf{p}}_r(\varphi_j) + \mathbf{d}(\Delta T) \\ \mathbf{q}_r(t_j) &= \mathbf{q}_r(t_{j-1}) * \widehat{\mathbf{q}}_r(\varphi_{j-1})^{-1} * \widehat{\mathbf{q}}_r(\varphi_j) * Rot(\tfrac{\omega_j + \omega_{j-1}}{2}\Delta T) \end{aligned} \tag{2.2}$$

where ω_j is the angular speed at time t_j.

The function $\mathbf{d}(\Delta T)$ approximates the translation component of a curved trajectory. Assuming that the ΔT is small enough to consider the parameter variation between t_{j-1} and t_j as linear, the translation approximation can be written as

$$\mathbf{d}(\Delta T) = \begin{pmatrix} \sin(\alpha_{j-1} + \frac{\omega_j + \omega_{j-1}}{2}\Delta T)\frac{||\mathbf{v_j}||+||\mathbf{v_{j-1}}||}{2}\Delta T \\ 0 \\ \cos(\alpha_{j-1} + \frac{\omega_j + \omega_{j-1}}{2}\Delta T)\frac{||\mathbf{v_j}||+||\mathbf{v_{j-1}}||}{2}\Delta T \end{pmatrix} \tag{2.3}$$

where $\mathbf{v_j}$ and α_j are the linear velocity, respectively the yaw angle at time t_j. In our coordinate system, a null yaw angle coincides with a forward locomotion direction along the world Z-axis. Fig. 2.3 illustrates the approximated root translation (the red line) of an original curved motion (the green curve).

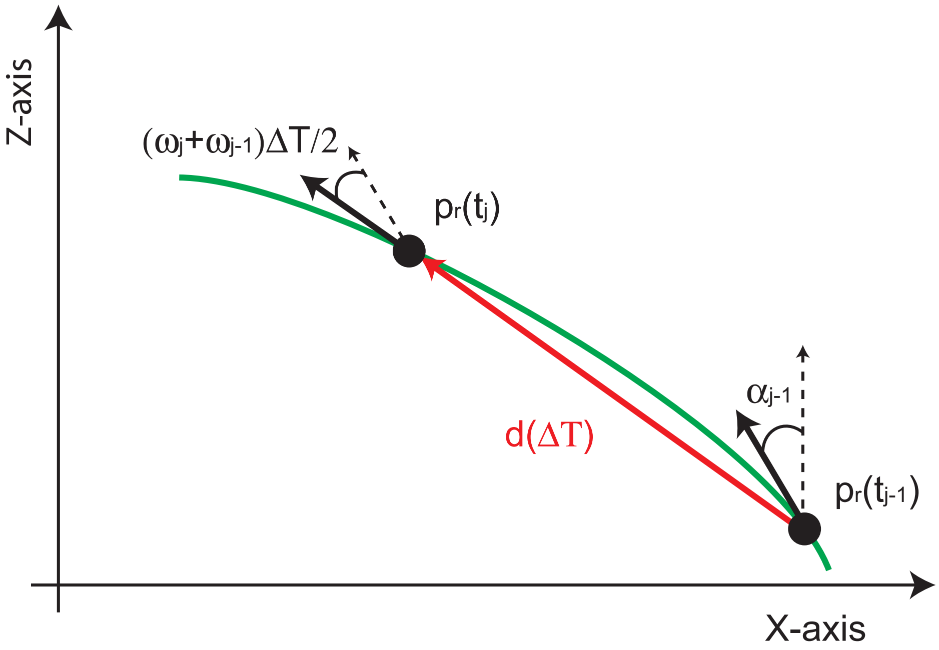


Figure 2.3: Top-view of the original root trajectory (green curve) approximated by the function $\mathbf{d}(\Delta T)$, illustrated by the red line, for a given $\Delta T = t_j - t_{j-1}$.

In practice, the anticipation of the local orientation of a given joint is not sufficient. We also need to know its position in the global coordinate system, in particular for foot joints in order to compute their Euclidean distance to the floor. With this aim in mind, the human body hierarchy has to be traversed, starting from the root node until the desired node, by multiplying every local node orientation. This operation is expensive, and it severely limits the number of anticipated frames given a continuous varying real-time animation context.

In short, given a controlled parameter variation (e.g. characterized by a set of parameter values to be reached within a desired period of time), the proposed method is able to anticipate postures and the Cartesian location of body segments in a global coordinate system. This material is used for footplant detection and enforcement, as explained in the next two sections.

2.3 Footplant Detection

A virtual human's activity can be segmented into parts having specific constraint types, allowing a formal description of its motion. One of the most important constraint type is the footplant, defined as a period of time during which a foot or part thereof (e.g. ankle, toe) remains in a fixed position with respect to the ground. We consider a footplant with two joints: the ankle (or heel) and the metatarsal (or toe) joints, according to the standard H-ANIM [H-ANIM, 2005] skeleton. The knowledge of constraint information is crucial in many situations. For example, it allows either to structurally align motions [Rose et al., 1998; Park et al., 2002a] or to correct artifacts such as footskate [Kovar et al., 2002b]. A footplant has to be detected with methods which have to return precise results. Actually, a too short footplant duration would not totally correct the foot sliding. Conversely, too long duration would stretch the leg when reaching the joint limits, resulting in motion discontinuities.

2.3.1 Thresholds on Cartesian Position and Speed

The detection of a footplant can be solved by using a standard method based on the current vertical position (or height) and translation speed of the foot, as described in [Lee et al., 2002; Menardais et al., 2004]. At each animation frame, foot constraints are checked. The foot is considered to be in contact with the ground when its position and linear speed are lower than specific thresholds.

However, this approach is not reliable for noisy or badly calibrated motion capture data. Fig. 2.4 illustrates the vertical position and speed values of a foot in function of time, for noisy data. Here, the determination of a position threshold ε_p is difficult. If the threshold value is too small (see threshold A on Fig. 2.4, left), the footplant is too short. By increasing the value, the footplant may be split into several pieces (see threshold B), or may be too long (see threshold C). This footplant splitting problem exists for the setting of the speed threshold ε_s as well (Fig. 2.4, right).

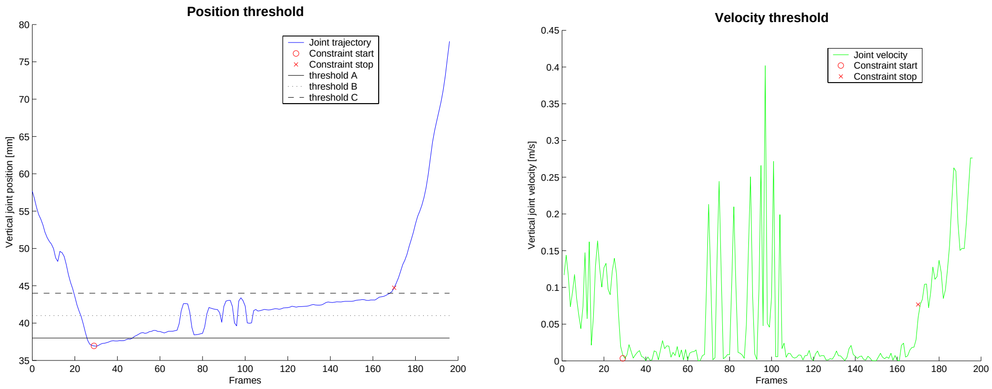


Figure 2.4: Problem of threshold values for footplant detection on noisy data: the correct footplant start frame (circle) and end frame (cross). **Left:** The vertical trajectory of the foot. **Right:** The speed curve of the foot.

To tackle this problem, the computed joint position at frame i may be replaced by an average of the $n+1$ positions from frame $i-\frac{n}{2}$ to frame $i+\frac{n}{2}$, using our anticipation method for future frames. In this way, the influence of peaks in noisy data is reduced. However, the difficulty remains to determine an appropriate n value. Our experiments have shown that this method fails to detect short footplant duration when increasing n.

In addition to the footplant splitting problem, the ε_p and ε_s thresholds are dependent on the input motions, due to their quality or their characteristics. During a single motion capture sequence, the quality may change, for example when one or more foot markers have been clumsily moved by the performer. Hence, the reference floor level moves up or down, entailing the need to modify ε_p. Different motion characteristics also result in adapting the ε_s threshold. For instance, Fig. 2.5 depicts the ankle speed over time, for a fast running and a slow walking motion. If a unique ε_s is used, the footplant detection fails at least for one of the two motions. In addition, an appropriate threshold for the heel is not necessarily valid for the toe.

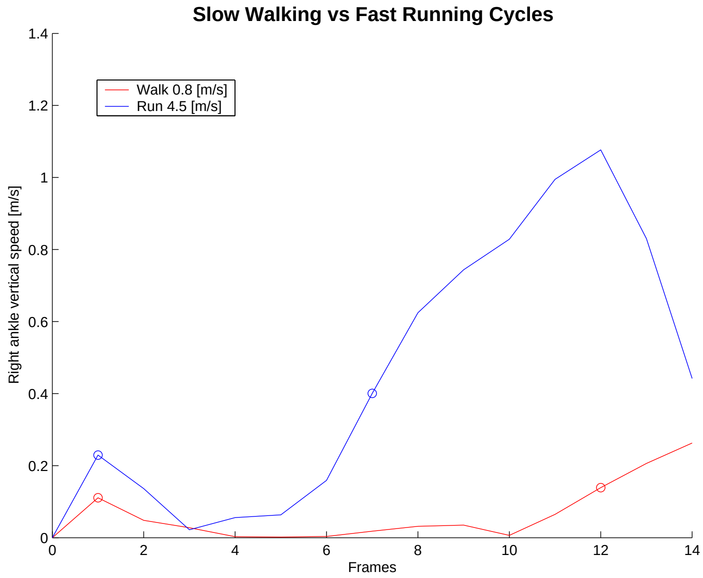


Figure 2.5: Right ankle linear speed value comparison between a slow walking (red) and a fast running (blue) motion. The circles indicate the start and end of the footplant.

One solution consists in determining an empiric threshold function, whose given motion characteristics return the corresponding ε_p and ε_s values. However, this approach is unpractical due to the significant number of parameters influencing the thresholds: motion speed; motion capture quality; performer style; joint type.

2.3.2 Adaptive Positional Threshold

We overcome these difficulties by detecting footplants on locomotion patterns in an on-line manner, using our motion anticipation technique. Roughly speaking, our method uses a vertical position (height) threshold only. This latter is adaptively computed by investigating a specific fixed set of frames of the input patterns.

We discard the speed threshold, which allows avoiding its problematic setting and improving the detection method efficiency. In fact, when the foot position is under ε_p, the speed threshold allows to identify unconstrained foot motion above the floor. This occurs typically during the swing phase, when the constrained foot leaves the floor and goes forward for the next step. Therefore, we aim at defining, in our patterns, a period of time during which we ensure that the foot has to be fixed to the ground if its position is under ε_p.

We make use of the observations performed on time-normalized walking and running patterns. From them, we assert that for a given type of locomotion (walk or run), a foot joint constraint occurs *roughly* inside a fixed time interval, regardless of other motion characteristics. Therefore, our method investigates this time interval, referred to as the enclosed frames defined from $\mathcal{F}_b$ to $\mathcal{F}_e$, in order to detect the correct frame $\mathcal{F}'_b$, at which the constraint starts, and $\mathcal{F}'_e$, at which the constraint ends (see Fig. 2.6). Those frames have a corresponding index computed with the $\text{idx}(\mathcal{F})$ function which returns the F frame index of the given frame $\mathcal{F}$. Hence, the frame index of $\mathcal{F}'_b$ is $F'_b = \text{idx}(\mathcal{F}'_b)$.

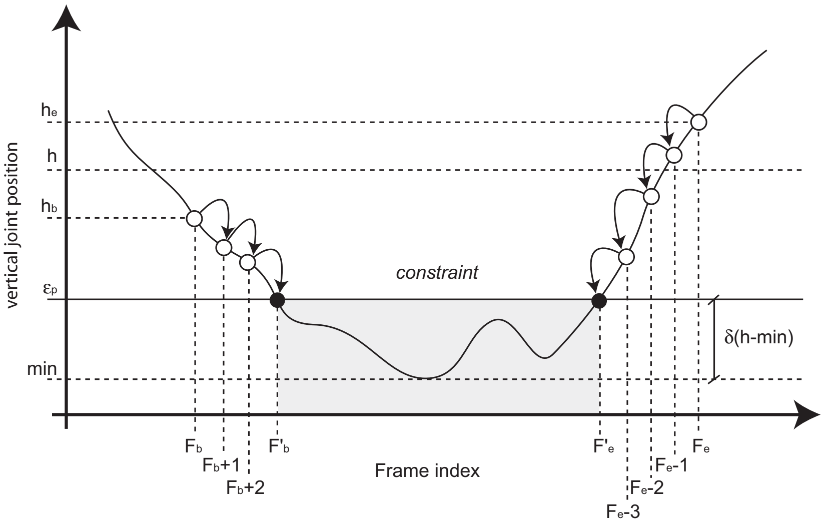


Figure 2.6: Schematic representation of the constraint detection method, with its parameters. The arrows indicate the successive comparisons between the vertical joint position (at a given frame) and the threshold value ε_p, in order to detect the correct frames at constraint start and end.

Instead of using a fixed threshold value, ε_p is computed adaptively according to the current $\boldsymbol{\theta}(\boldsymbol{\Psi})$ motion unit. From this motion and for a given foot joint, we attach the set of frames $\mathcal{I} = (\mathcal{F}_b, \ldots, \mathcal{F}_e)$. Then, the global heights h_b and h_e of the given joint are extracted from the bounding frames $\mathcal{F}_b$ and $\mathcal{F}_e$ respectively. In order to determine ε_p adaptively, we assume that the mean h of these two values is related to the pattern properties, namely its locomotion speed and style. To confirm, we compare various h values computed for the right heel and left toe, at different locomotion speeds and for 5 subject capture styles. Fig. 2.7 illustrates the results for walking and Fig. 2.8 for running motion units, where each cell contains the corresponding mean height, in meters.

From these pictures, we infer that h is dependent on the motion parameters and therefore that the adaptive ε_p is a function of h. Clearly, the ε_p threshold has to be greater than the min minimal vertical position on $\mathcal{I}$, but smaller than h, as described in Eq. 2.4. We introduce the

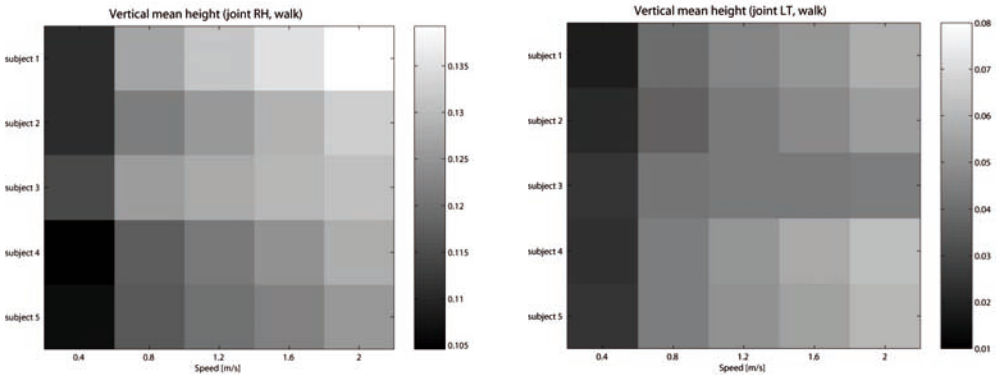


Figure 2.7: Joint's mean height (in meter) for bounding frames $\mathcal{F}_b$ and $\mathcal{F}_e$ for walking motion units at various speeds. **Left:** Right heel comparison. **Right:** Left toe comparison.

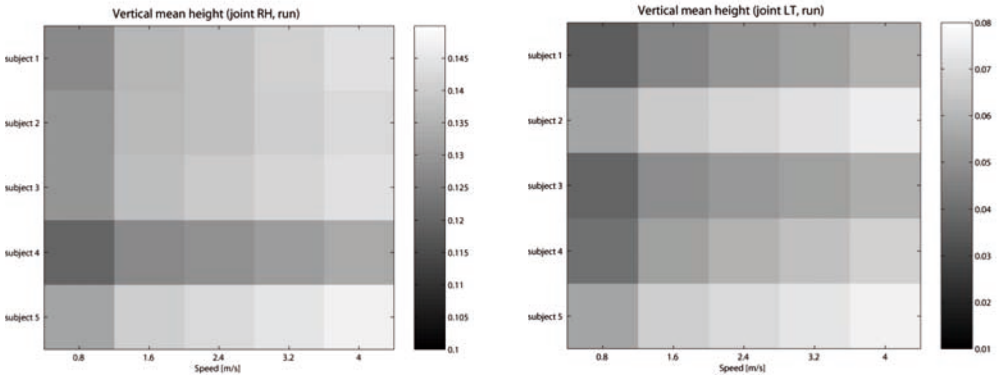


Figure 2.8: Joint's mean height (in meter) for bounding frames $\mathcal{F}_b$ and $\mathcal{F}_e$ for running motion units at various speeds. **Left:** Right heel comparison. **Right:** Left toe comparison.

δ constant defined in $[0 \ldots 1]$ characterizing the correctness of the default bounding frame indices $\mathrm{idx}(\mathcal{F}_b)$ and $\mathrm{idx}(\mathcal{F}_e)$. As an example, by setting δ close to 1, it means that $\mathrm{idx}(\mathcal{F}_b)$ and $\mathrm{idx}(\mathcal{F}'_b)$ are close to each other. Fig. 2.6 schematizes the method parameters.

$$\varepsilon_p = min + \delta(h - min) \tag{2.4}$$

The detection algorithm is described in Algo 3. In practice, the start frame index F'_b of a footplant, for a given joint, is detected as follows. We start by comparing the joint's vertical position at frame $\mathcal{F}_b$ (left circle on curves in Fig. 2.9) with the ε_p (dashed line in Fig. 2.9) computed in Eq. 2.4. If the position at frame $\mathcal{F}_b$ is already below the threshold (e.g. the curve representing a 4.5 [m/s] run on Fig. 2.9, bottom), $\mathrm{idx}(\mathcal{F}_b)$ is assigned to F'_b. Otherwise, as long as position is above the threshold, the comparison is shifted to the next frame. When the comparison fulfills the threshold condition, the current frame index is assign to F'_b. Analogously, F'_e is determined by starting at frame $\mathcal{F}_e$. The threshold comparison is shifted to the previous frame as long as the current frame value stays above the threshold. In this way, our method ensures that a footplant never splits off into small pieces.

Algorithm 3 Constraint detection algorithm for the joint jnt, and given enclosed frames $\mathcal{F}_b$ and $\mathcal{F}_e$. The algorithm returns the correct start and end constraint frame indices, F'_b and F'_e respectively.

```
{vPos($jnt$, $\mathcal{F}$) returns the vertical position of the joint $jnt$ in frame $\mathcal{F}$}
$h_b$ := vPos($jnt$, $\mathcal{F}_b$)
$h_e$ := vPos($jnt$, $\mathcal{F}_e$)
$h$ := ($h_b$ + $h_e$)/2
{minVPos($jnt$, $\mathcal{I}$) returns the minimal vertical position of $jnt$ over all frames in $\mathcal{I}$}
$min$ := minVPos($jnt$, $\mathcal{I}$)
$\varepsilon_p$ := $min$ + $\delta$ ($h$-$min$)
{Compute the constraint start}
$\mathcal{F}'_b$ := $\mathcal{F}_b$
$p$ := vPos($jnt$, $\mathcal{F}'_b$)
while $p > \varepsilon_p$ do
  $\mathcal{F}'_b$ := next($\mathcal{F}'_b$)
  $p$ := vPos($jnt$, $\mathcal{F}'_b$)
end while
$F'_b$ := idx($\mathcal{F}'_b$)
{Compute the constraint end}
$\mathcal{F}'_e$ := $\mathcal{F}_e$
$p$ := vPos($jnt$, $\mathcal{F}'_e$)
while $p > \varepsilon_p$ do
  $\mathcal{F}'_e$ := prev($\mathcal{F}'_e$)
  $p$ := vPos($jnt$, $\mathcal{F}'_e$)
end while
$F'_e$ := idx($\mathcal{F}'_e$)
```

2.3.3 On-line Detection

At first sight, this detection method seems inappropriate for an on-line context. The movement needs to be known in advance to obtain h_b and h_e, in order to determine the threshold ε_p. Thanks to the anticipation, each frame $\mathcal{F}_i$ of motion unit $\boldsymbol{\theta}(\boldsymbol{\Psi})$ can be used at any time. Therefore, when a new $\boldsymbol{\theta}(\boldsymbol{\Psi})$ is generated, the detection method is applied on the four joints describing both feet, providing a start and end frame index per constraint in the time-normalized pattern.

In case of a continuous modification of $\boldsymbol{\Psi}$, the detection algorithm is executed at each time step. This is necessary, as $\boldsymbol{\theta}(\boldsymbol{\Psi})$ changes continuously.

2.4 Footplant Enforcement

We address the problem of re-positioning and enforcing a footplant by using the numerical IK algorithm incorporating the priorities presented in [Baerlocher and Boulic, 2004]. This method ensures that at least the high priority end-effector goal is achieved at best before

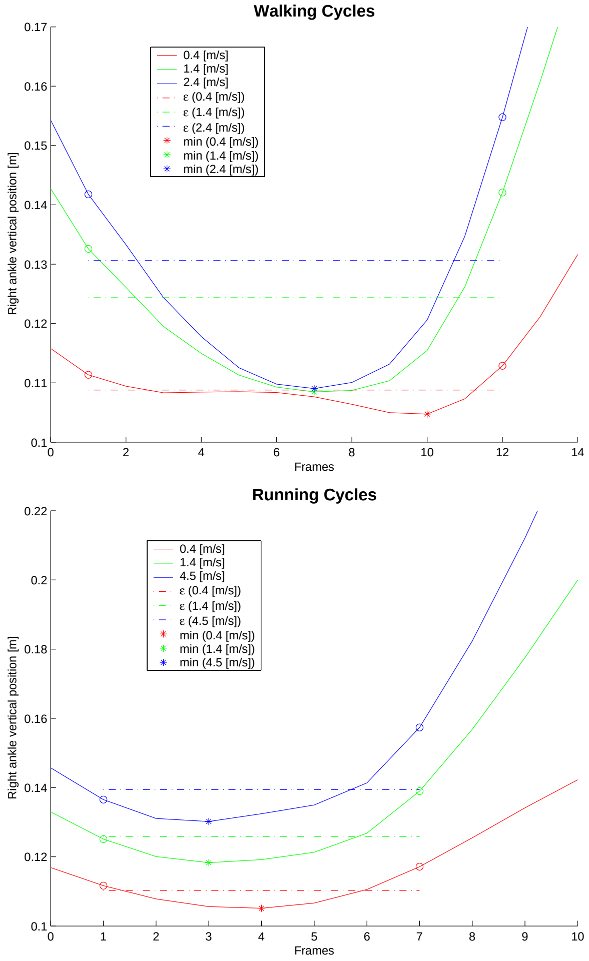


Figure 2.9: Examples of our motion data for the right ankle vertical position. The circles indicate the bounding frames. **Top:** different speed values for walking. **Bottom:** different speed values for running.

considering lower priority constraints.

2.4.1 Inverse Kinematics Model

We attach two positional end-effectors on each foot, one for the heel and the other for the toe. In order to limit the computation time of the IK algorithm, we simplify the IK configuration by defining two independent IK chains, one for each leg. A chain contains six DOFs, from the hip joint to the toe joint.

Basically our footplant enforcement method works as follows. In an unconstrained state, the effector attached to a foot joint is driven by the locomotion pattern. As soon as its corresponding constraint has to be applied, the effector goal is re-positioned on the floor and remains fixed during the whole constraint duration. To ensure motion continuity, the effector trajectory is smoothed around the constraint, by defining ease-in and ease-out phases. We therefore apply the displacement map technique [Bruderlin and Williams, 1995; Witkin and

Popović, 1995] by describing a displacement map $\mathbf{d}(t)$ in order for the corrected motion $\boldsymbol{M}_{cor}(t) = \boldsymbol{M}(t) \oplus \mathbf{d}(t)$ to satisfy the detected footplants.

Let $\mathcal{C}$ be a constraint defining a position goal $\mathcal{C}_p$ for the effector E, active from t_1 to t_2. The desired effector trajectory $E(t)$ is built from its original trajectory $E_0(t)$ as:

$$E(t) = \begin{cases} E_0(t) + \left(1 - \Gamma(\frac{t-t_1+\sigma}{\sigma})\right)(\mathcal{C}_p - E_0(t_1)) & \text{for } t_1 - \sigma \leqslant t < t_1 \\ \mathcal{C}_p & \text{for } t_1 \leqslant t \leqslant t_2 \\ E_0(t) + \Gamma(\frac{t-t_2}{\sigma})(\mathcal{C}_p - E_0(t_2)) & \text{for } t_2 < t \leqslant t_2 + \sigma \\ E_0(t) & \text{otherwise} \end{cases} \tag{2.5}$$

where σ is the ease-in and ease-out duration. The Γ function is the descending cubic step function, described in Eq. 2.6 and plotted in Fig. 2.10.

$$\Gamma(t) = 2t^3 - 3t^2 + 1 \quad \text{for } 0 \leqslant t \leqslant 1 \tag{2.6}$$

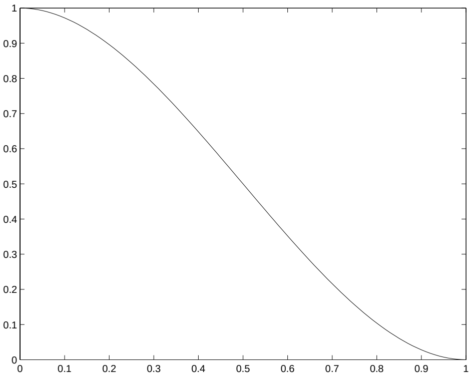


Figure 2.10: The descending cubic step function used for the ease-in/out phases.

Similarly to Lee et al. [1999], we consider a motion as a set of independent character postures. At each time $t = t_i$, the original posture $\boldsymbol{M}(t_i)$ is modified by applying IK with end-effector trajectories defined in Eq. 2.5. Fig. 2.11 illustrates the original and corrected vertical trajectories of the two effectors attached to the right foot. In addition, we choose to set highest priorities on the ankle's effectors: their constraints occur first and have significant visual impact. The IK solver therefore ensures that, at least, the ankle constraint is enforced, before trying to enforce the toe constraint. This effect is visible in Fig. 2.11 (right, between frames 24 and 28, and between frames 55 and 59), as the corrected toe position trajectory goes down just after the release of its constraint (position error under than 0.01 meters). At this instant, the IK algorithm is not able to compute a solution ensuring a correct position for both effectors, due to an important foot sliding observed in the noisy input data. However, thanks to the priorities, the position of the ankle joint is perfectly corrected (Fig. 2.11, left).

2.4.2 Ease-in with Anticipation

The end-effector trajectories are constructed in order to ensure smooth motion generation around the constraint. Examining Eq. 2.5, one can observe that while performing the ease-in

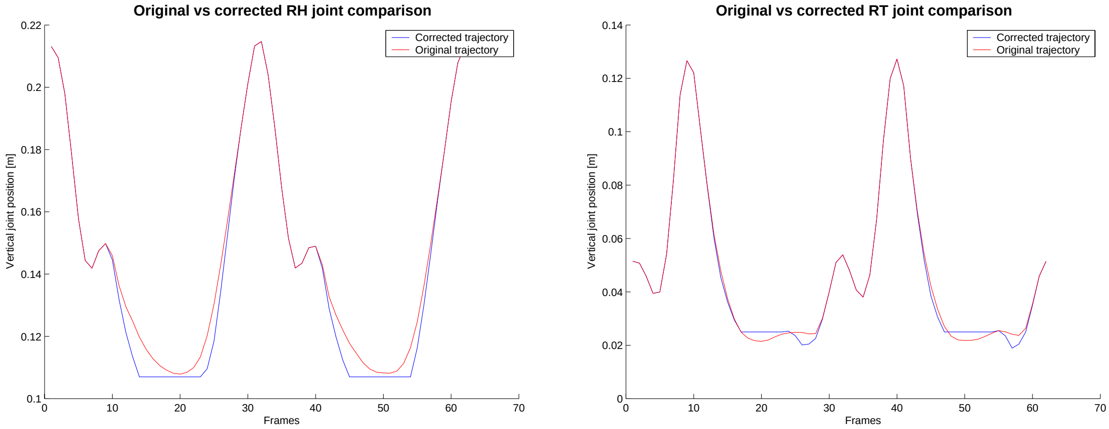


Figure 2.11: Original $E_0(t)$ and corrected $E(t)$ vertical effector trajectories for the right foot.**Left:** Effector attached to the ankle. **Right:** Effector attached to the toe.

phase, the original position of the effector E_0 at (the future) time t_1 is required.

Thanks to the anticipation, this position and its corresponding constrained position $\mathcal{C}_p$ are computed as follows. At the beginning of the ease-in phase, namely at time $t_1 - \sigma$, the anticipated posture $\boldsymbol{M}(t_1)$ is computed to extract $E_0(t_1)$. This position, which corresponds to the original one of its attached foot joint, is re-positioned to be on the ground and stored in $\mathcal{C}_p$. Fig. 2.12 illustrates the current posture (yellow) at time $t - \sigma$ and the anticipated one (green) at time t_1 for each effector.

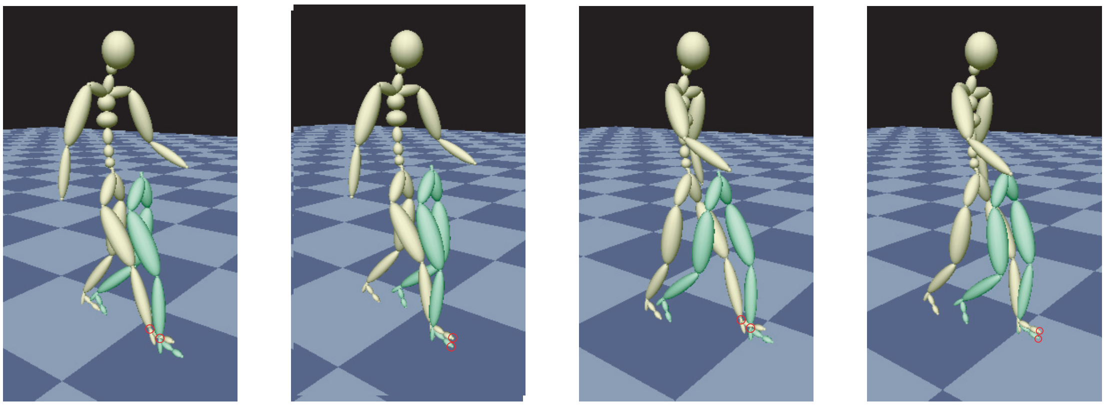


Figure 2.12: The start of an ease-in phase for each effector. Current (yellow) and anticipated postures (green) are computed. **From left to right:** Effectors are attached to: right ankle, right metatarsal, left ankle and left metatarsal.

2.4.3 Ease-in Activation

Before enforcing a footplant, we have to know its start time t_1 in order to activate its ease-in process at time $t_1 - \sigma$. This is performed using our footplant detection method, applied differently according to three scenarios: constant motion parameter, planned motion parameter variation and modification of a planned motion parameter variation.

2.4.3.1 Constant motion parameter

In this first case, the motion parameter vector $\mathbf{\Psi}$ is constant over time. At the initialization stage, triggered at time t_i (when $\mathbf{\Psi}$ is set), a new cycle $\boldsymbol{\theta}(\mathbf{\Psi})$ is computed and footplants are detected with our method. To systematically check if there is already a footplant to activate, the $[t_i \dots t_i + \sigma]$ time interval is regularly sampled into n Δt intervals. For each j-th interval, the time $t_i + j\Delta t$ is used to compute its corresponding locomotion phase φ_j. This value is multiplied by the frame number N_{frame} of $\boldsymbol{\theta}(\mathbf{\Psi})$ to obtain the corresponding idx$(\mathcal{F}_j)$ frame index in the normalized time. If this frame index is more than or equal to the F'_b of a foot joint, a constraint is et for time $t_i + j\Delta t$. As $t_i + j\Delta t < \sigma$, the ease-in phase is immediately activated.

After this initialization stage, at any t time, the future $t + \sigma$ time is transformed into the normalized time to detect if a constraint needs to be activated. This is necessary as the initialization stage does not ensure that all constraints have been detected. In fact, only those over the time interval $[t_i \dots t_i + \sigma]$ have been considered.

2.4.3.2 Planned motion parameter variation

In this case, a new $\mathbf{\Psi}'$ has to be reached within a given duration Δm, involving a controlled change of $\mathbf{\Psi}'$ over time (linear in our case). We therefore assign a $\mathbf{\Psi}'_j$ parameter vector to each future time $t_i + j\Delta t$, for which a new $\boldsymbol{\theta}(\mathbf{\Psi}'_j)$ locomotion cycle is generated and footplants are detected. Then, analogously to the previous case, the corresponding idx$(\mathcal{F}_i)$ frame index for time $t_i + j\Delta t$ is extracted and compared to F'_b. In practice, refreshing the constraint boundaries at each time step is currently too expensive for real-time. Hence, frame indexes F'_b and F'_e are pre-computed by applying the detection method on motions with various parameter vector (7 and 11 different normalized speed values for walking, respectively running cycles). Theses frame indexes are then used to determine the ones of $\boldsymbol{\theta}(\mathbf{\Psi}'_j)$ by applying linear interpolation. Our results confirm the pertinence of this approach as can be seen in Section 2.5 of this chapter.

2.4.3.3 Modification of a planned motion parameter variation

This last case handles the situation in which a planned variation is interrupted by a new one at time t'. For each future $t_i + j\Delta t$ time, a new locomotion pattern is computed and footplants are detected, similarly to the previous case. However, if an effector is in an ease-in phase when the variation is interrupted, its trajectory has to be carefully modified, as its goal position $\mathcal{C}_p$ and time t_1 have changed, due to the new motion parameter configuration.

Fig. 2.13 illustrates this modification schematically. First, the effector trajectory $E_0(t)$, depicted by the blue line, is continuously modified to reach the $\mathcal{C}_p$ constraint. This creates a new trajectory $E(t)$, depicted by the dashed blue line. Due to the motion parameter changes at a time t' during the ease-in phase, a new $E'_0(t)$ original effector trajectory is provided (green curve). We have therefore to modify $E(t)$ in order to reach the new goal position $\mathcal{C}'_p$ at a new time t'_1. We modify Eq. 2.5 in order for the effector trajectory, modified from $E'_0(t)$ to remain smooth. Therefore, from time t' the effector trajectory, depicted by the red dashed line, is updated as follows:

$$E(t) = E_0'(t) + \left(1 - \Gamma\left(\frac{t-t'}{t_1'-t'}\right)\right)\left(E_0'(t') - E(t')\right) + \Gamma\left(\frac{t-t'}{t_1'-t'}\right)\left(\mathcal{C}_p' - E_0'(t_1')\right) \quad (2.7)$$

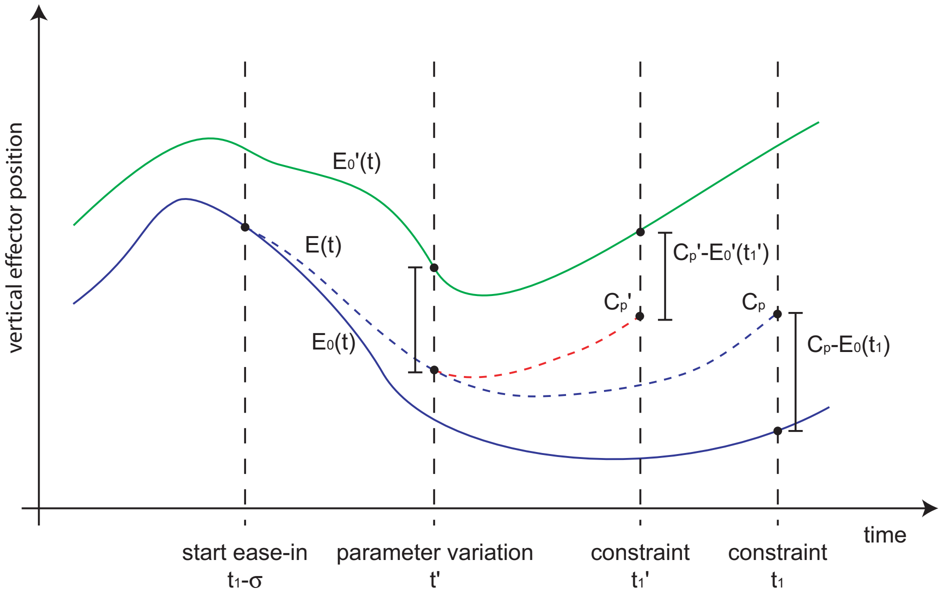


Figure 2.13: Step variation during the ease-in phase. The blue and the green curves correspond to the original effector trajectories, the dashed blue and red curves to the modified one.

Fig. 2.14 illustrates the smoothness of the modified effector trajectories. The animation is generated first by continuously increasing the walking speed. This process is then interrupted by another parameter variation, consisting in a transition to a running motion.

2.5 Results

First we describe the different components of our system prior to presenting some on-line application examples.

2.5.1 Putting it all together

For our experiments, we used the motion generation method described in the previous chapter to generate the locomotion units. Those are composed of 25 frames and animate an H-ANIM body with 40 DOFs. To speed up the motion anticipation computation, only the lower body postures are computed. All the 25 posture configurations, including the joint global positions, are calculated within 1.3 ms.

Then the enclosed frame set $\mathcal{I}$ has to be determined, for walking and running cycles, and for each joint needing floor contact detection. We use the values summarized in Table 2.1.

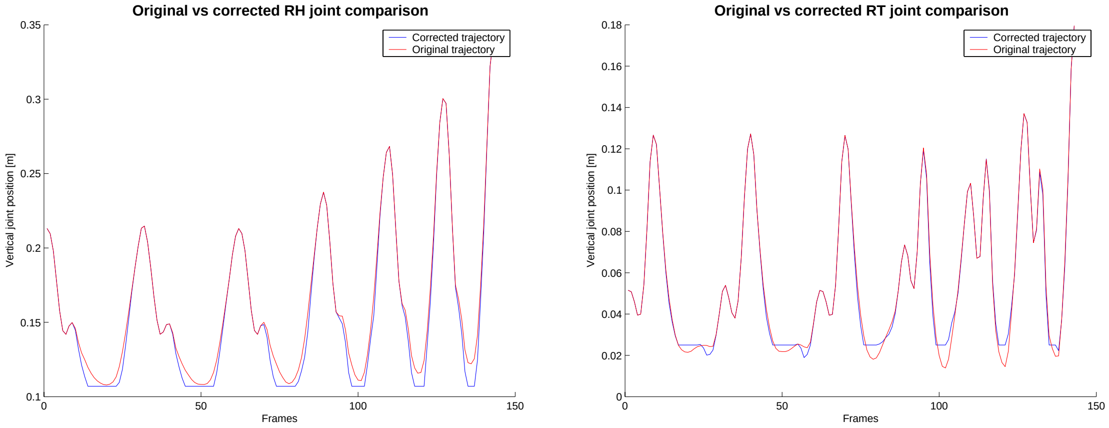


Figure 2.14: Original $E_0(t)$ and corrected $E(t)$ vertical effector trajectories for the right foot. The animation first describes a walk which continuously speeds up before being interrupted by a continuous transition to a run. **Left:** Effector attached to the ankle. **Right:** Effector attached to the toe.

These frames' indexes define the interval in which the constraint detection algorithm is performed. In our experiments, δ is equal to 0.4, and the exact starting and ending constrained index frames F'_b and F'_e for all joints, are computed in less than 0.05 ms.

Joint's name	**idx**$(\mathcal{F}_b)$ **(walk)**	**idx**$(\mathcal{F}_e)$ **(walk)**	**idx**$(\mathcal{F}_b)$ **(run)**	**idx**$(\mathcal{F}_e)$ **(run)**
Right ankle	1	12	1	7
Right metatarsal	3	15	1	9
Left ankle	13	26 (mod 25)	12	19
Left metatarsal	15	29 (mod 25)	13	22

Table 2.1: Frames interval for each constrained joint, observed on 25 frames locomotion cycles

After detecting the constraint relative to an effector, the method continuously checks whether a constraint has to be enforced in a near future, in order to activate the ease-in phase. The computation time for footplant enforcement depends on the number of joints whose trajectory is modified by the IK algorithm. On average, 1.8 ms are necessary to smoothly enforce the two footplants, representing four constraints.

Finally, we have tested our method with a continuous motion parameters variation and we obtained a maximal computational cost of 4.7 ms per frame. This is the most expensive case, as it entails pattern computation for each future time $t_i + j\Delta t$.

2.5.2 Foot Sliding Correction

The first experiment consists in cleaning up the foot sliding and penetration into the ground. Our original motion shows some artifacts, as illustrated on the top row of Fig. 2.15. This is due to the input motion capture data and the PCA algorithm used to reduce data dimensionality. The footplants are then correctly detected and re-positioned, no more foot sliding is perceptible and IK does not introduce discontinuities during the animation.

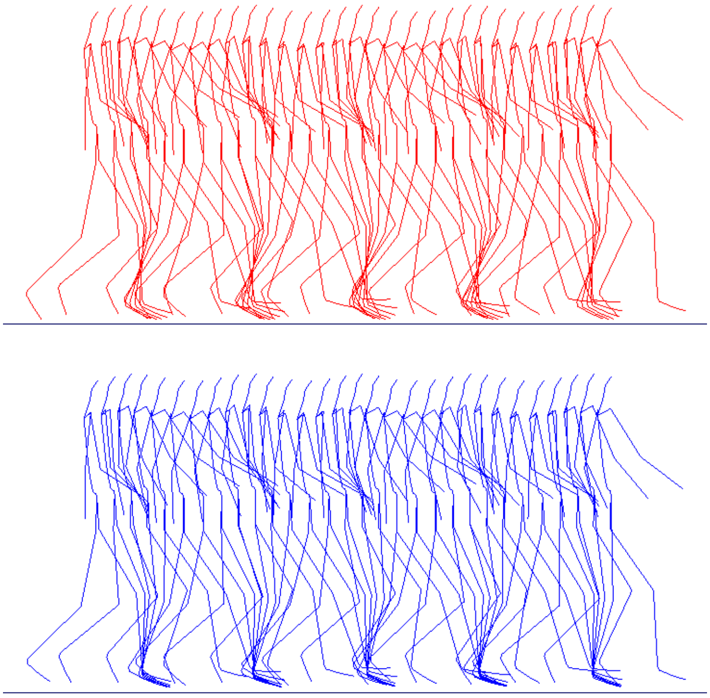


Figure 2.15: Foot sliding clean up. **Top:** Original motion. **Bottom:** Modified motion.

2.5.3 Stylistic Edition

Another application of our methodology is to create small stylistic variations of the generated motion, by modifying the location of one (or more) end-effector during its constraint. In our example, the right toe constrained position is displaced, as depicted in Fig. 2.16, to produce a pigeon-toed effect. At the beginning of the ease-in phase, we modify the anticipated toe joint position corresponding to the start of the constraint. This modification consists in a rotation, having its center placed at the right ankle joint, around a perpendicular axis to the ground. The rotation angle is set in order to direct the toe towards the left ankle.

2.5.4 Continuous Parameter Variation

The on-line reactivity to user parameter modification is an important aspect of our method. At any time, the user or the Artificial Intelligence (AI) driving the autonomous agent can define a new motion parameter set that has to be reached in a given period of time. The smooth parameter evolution is performed by a controller. In our implementation, we control the parameter evolution linearly. Let us imagine a slow walking pattern. By setting a higher speed value and changing the locomotion from walk to run, the motion will simultaneously

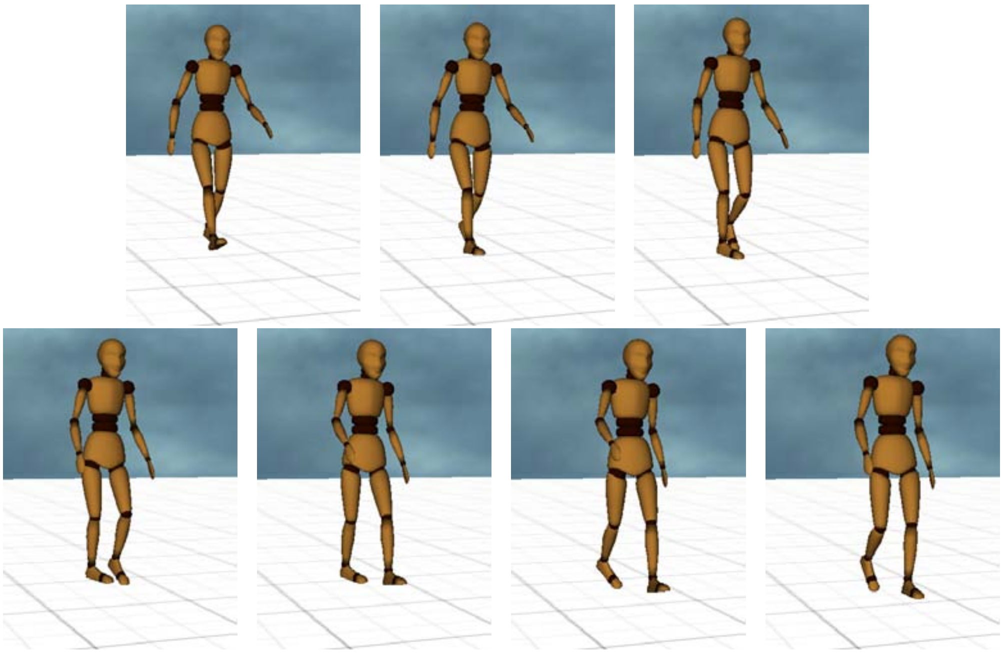


Figure 2.16: Modification of the right toe position produces a right pigeon-toed walking motion.

accelerate and start to run (see Fig. 2.17). Therefore, the motion is continuously modified while guaranteeing that the footplants detection and enforcement are still coherent with the corresponding parameter values.

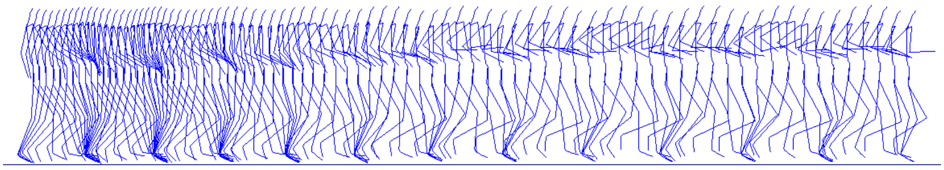


Figure 2.17: Continuous parameter variation, from a walk at 0.8 m/s to a run at 1.9 m/s (parameter variation within 3 sec).

2.5.5 Curved Path

In the last experiment, our straight-line locomotion cycles are adapted so as to produce a curved path, illustrated in Fig. 2.18. The motion angular speed is continuously changed and therefore modifies the yaw angle of the root node. Thanks to our method, the foot remains fixed to the floor during the constraint, while the root rotates. When the constraint is relaxed, the ankle first smoothly reaches its original trajectory. The toe is fixed to the ground a little longer, allowing it to rotate around its position.

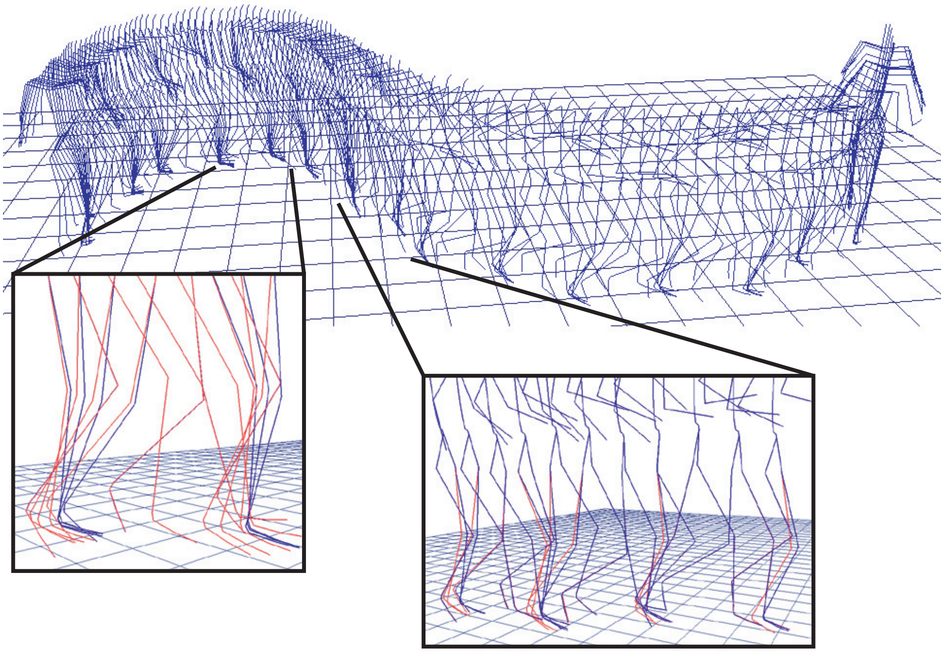


Figure 2.18: Walking to running with angular speed variation. The red stick figures in the zoomed frames correspond to the original motion.

2.6 Conclusion

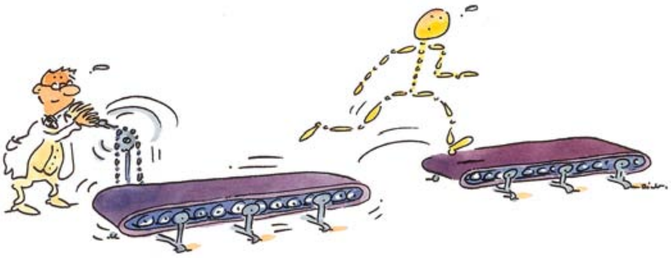
To correct a keyframe animation, an animator has to detect footplants manually by labeling the constrained frames. Then the enforcement of these constraints is performed using methods either based on numerical or analytical inverse kinematics. We improve this process by presenting an on-line animation system based on the generation of anticipated postures to detect, re-position and apply footplants. This system enhances not only the motion quality by correcting foot sliding, but also the flexibility of our locomotion engine by considering also the angular speed parameter.

The detection algorithm is robust because it exploits the properties of the motion (e.g. speed, type of locomotion, human size) from which the foot constraints have to be extracted on-line. For a given motion, we define adaptively only one vertical position threshold value for each joint describing a foot. The threshold intrinsic normalization allows the method to work for any human size. A numerical IK solver with priorities is applied to re-position and enforce a footplant, described by two end-effectors. The priorities set on these end-effectors ensure to exactly reach at least one goal position. Our method maintains smooth end-effectors trajectories by anticipating postures at a constraint.

The generated motion continuity has also been evaluated. We can modify the goal position for a given footplant, allowing to introduce some smooth stylistic variations in the original motion. We also modify the root yaw orientation to perform curved motion, leading to very satisfying results. The coherence of the method is ensured, as the motion parameters vary continuously. Finally, the method is computationally tractable as even in the worst situation, where parameters have to be updated at each time step, a frame update is performed in less than 5 ms.

A limitation of our method concerns the IK method, which is per-frame based. Theoretically discontinuities may occur, as the current modified frame is computed relatively to the original one. Therefore, it could happen that the IK solver finds very different resulting postures for two original consecutive frames, leading to discontinuities between the modified frames. In practice we never observed motion discontinuities during our experiments. An alternate approach would be to compute the current modified frame by applying IK to the previously modified one.

In our example, the curved motion generation may produce important end-effector position modifications in the case of important angular speed with low linear speed. In such an extreme situation the constraint duration of the foot on the outer curve could be reduced.



Chapter 3

Coherent Transition Generation between Locomotion and Jump

3.1 Introduction

The animation engine presented and detailed in the Chapter 1 (Part III) allows to generate two classes of motion: the locomotion, characterized by a continuous stream (cyclic) of parameterized walking or running patterns, and the jumping motion, characterized by independent sequences (episodic) of parameterized jumps. However, an animator may need to join a walking action with a running jump, so that a character could clear an obstacle for example. The simplest solution consists in retrieving the most appropriate transition clip from a motion capture database [Kovar et al., 2002a; Arikan and Forsyth, 2002; Lee et al., 2002]. The search criteria are based on the current motion parameters, such as the speed and the type of locomotion, but also on the requested jump parameters, such as the length. This approach has obviously two main drawbacks. First, the user's request does not necessary match exactly the motion found in the database, or even worse, can return any motion clip. The quality of the resulting query is improved as the motion number increases in the database. Still, it induces the second drawback: the computation time goes up as the best motion has to be chosen from more possible candidates.

In this chapter, the goal is to focus on a real-time method which generates the transitions (or blend) itself, without searching from a motion clip database. Keeping this idea in mind, it is important first to ensure the dynamic coherence between the blended motions. For example, once a character starts to jump, the motion can not be modified due to the ballistic phase. Hence, similar structures from the blended motions, referred to as support phases, have to be identified and aligned in time.

Traditional blending methods [Kovar and Gleicher, 2003; Rose et al., 1998; Menardais et al., 2004] assume that the footplants composing a support phase are already provided from manual identification. We propose therefore an improvement by providing two complementary footplant detection methods. The first one is based on the detection algorithm presented in the previous chapter of this thesis, adapted for locomotion. However, this method reaches its limits for jumping motions. In fact, various jumps contain more variety than various lo-

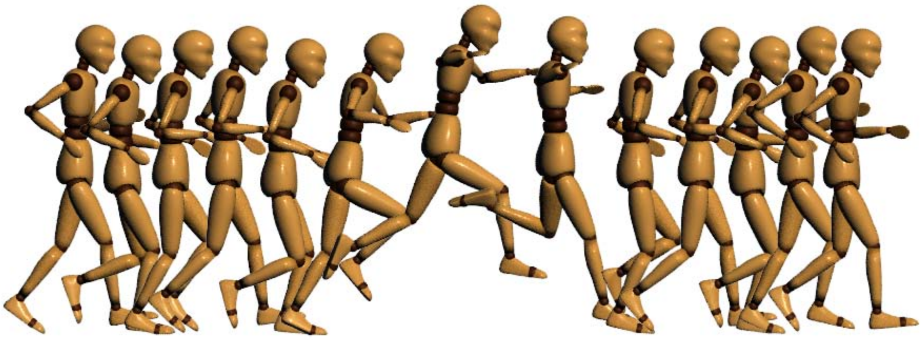


Figure 3.1: A transition generated by our method

comotion patterns. For that reason, another detection method is necessary. It is based on a semi-automatic technique which consists in manually pre-labeling the support phases of only a sample of parameterized motions. Owing to a relationship between these motion parameters and their support phases, the footplants of any new generated motion are detected on-the-fly. Finally, these support phases are dynamically aligned in time, as soon as the user requires a transition. We therefore stand out from static methods [Kovar and Gleicher, 2003; Rose et al., 1998] which align motions in a pre-processing stage.

From this support phase detection, the choice of a transition time (or instant) and duration is the second important aspect of a transition method. All of the works mentioned above perform blending as soon as it is requested by an animator. In some situations, the resulting produced motion is not coherent: the blending of two walking motions that are out-of-phase or the transition from a walk to a jump whose length does not match the run-up speed for example. This comes from either an incorrect chosen transition time and/or duration, or from a violation of dynamic motion properties. Hence, our method also controls the blending by choosing the appropriate transition time and duration, and by finding automatically a motion that minimizes dynamic incoherence.

To illustrate our method, we focus on the blending from cyclic to episodic motions. An episodic motion is played once, while cyclic motion is played repeatedly. We apply our locomotion engine, able to vary continuously its type between walking and running, with the two jumping engines: walking and running jumps.

Briefly explained, our methodology is decomposed into two stages. The off-line stage allows to identify the different support phases (e.g. single support with the left foot on the floor) during which the motions can be blended. The second stage is performed on-the-fly, driven by a state machine. As soon as a jump is requested, the method generates a run-up phase over a multi-dimensional parameter space, to best fit the different dynamic properties between the locomotion and the jump. Then the transition starting time is determined, by detecting automatically the correct support phase. The blending is performed during this phase. Finally, at the end of the jump, the locomotion parameters are smoothly restored to their initial values.

3.2 Method Overview

When a cyclic motion is played in real-time, an episodic motion may occur on the request of an animator. Therefore, a blending has to be executed, to generate a seamless transition from walking (running) to jumping and then back to walking (running). Without loss of generality, we assume in this chapter that a jump starts with the right take-off foot.

First we define eight foot events (see Table 3.1) which interact with the floor. These are needed to identify the different support phase types described in Fig. 3.2. A support phase is defined as a period of time during which one foot (single support) or two feet (double support) touch the ground. We simplify this definition by assuming that the heel touches always the floor before or simultaneously the toe, and conversely when the foot leaves the floor.

Table 3.1: Foot events interacting with the floor

Event name	abbreviation
Right heel strike	RHS
Right heel off	RHO
Right toe strike	RTS
Right toe off	RTO
Left heel strike	LHS
Left heel off	LHO
Left toe strike	LTS
Left toe off	LTO

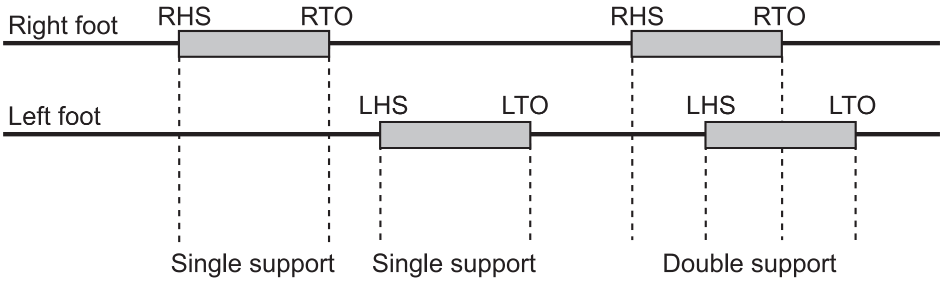


Figure 3.2: Description of the support phase types

To control the blending, we model a state machine, composed of six states (Fig. 3.3). The neutral state corresponds to the continuous cyclic locomotion, parameterized by the vector $\mathbf{\Psi}$. As soon as the animator or the application level (for example an autonomous agent) desires to perform a jump, he sets a parameter vector $\mathbf{\Psi}_{jump}$ to generated the corresponding jumping sequence. Then the model goes to the next state, where the speed s and locomotion weight w_{loco} from $\mathbf{\Psi}$ are adapted to match those from $\mathbf{\Psi}_{jump}$.

When the motion properties are compatible with the desired jump, the model starts the transition once the support phases of both motions contain similarities. However, it is not sufficient to compare double with single support phases as suggested by Menardais et al.

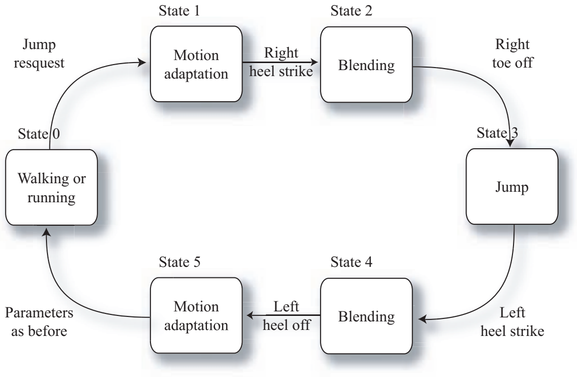


Figure 3.3: The state machine for controlled blending

in [2004]. Indeed, as illustrated in Fig. 3.4, the blend between a double support phase with the right foot backward (walking) and a single phase with the right foot forward (jump) produces incorrect result.

Therefore the blending starts only when the event RHS is detected (state 3). To ensure on-line performance, this detection is carried out using a method explained in Section 3.3.2. This blending is performed, from walking (or running) motion to jumping motion, until the RTO event. After that, the jumping motion is played until the LHS event is encountered, going to the next state. This state activates the blending back to the locomotion until the LHO. Finally, the locomotion engine returns to its neutral state, by modifying the motion parameters to the values they had before the jump request.

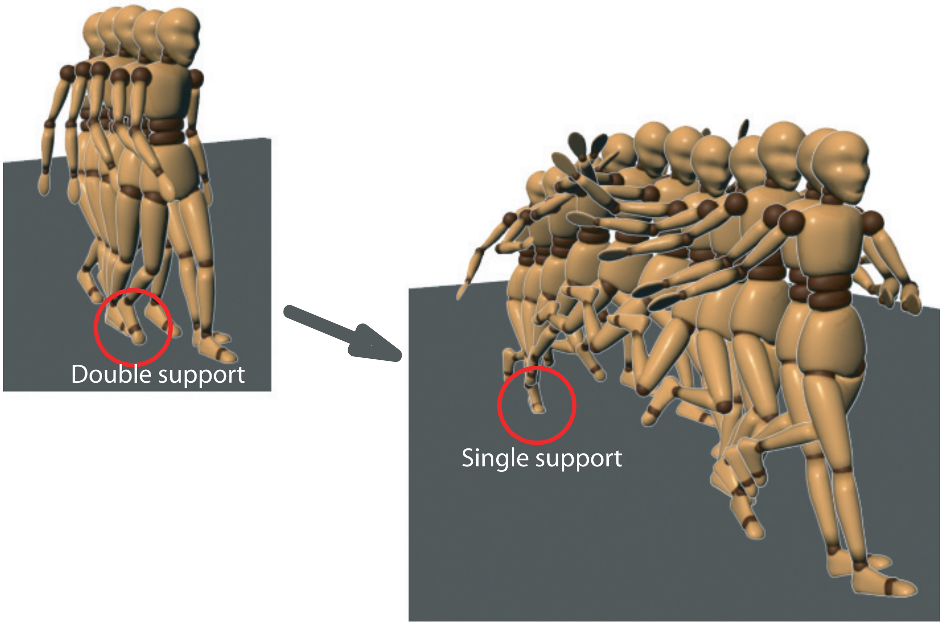


Figure 3.4: Incompatible support phases to blend a walk with a jump motion

3.3 Controlled Blending

We describe precisely the method to control a blending operation from locomotion to a jump, composed of three stages. Firstly the locomotion is adapted to generate a run-up phase compatible with the requested jump, ensuring a coherent transition. The second stage determines the transition time and duration automatically. Finally, the transition itself is performed by aligning in time the two blended motions.

3.3.1 Coherent Motion Adaptation

Imagine that a jump, described with a parameter $\mathbf{\Psi}_{jump}$, is requested at time t_1 while the character is animated by a locomotion cycle parameterized with $\mathbf{\Psi}_1$ whose its speed is s_1. Two properties have to be satisfied before effectively operating the transition: the final run-up speed s_2 to be reached before execution the jump at time t_2 and the run-up duration $t_{runup} = t_2 - t_1$.

3.3.1.1 Final Run-up Speed

In order to preserve the dynamic coherence, a jump of a specific length l can only be performed for a locomotion speed range. To determine this range, we analyze jumps from our motion capture database to establish a relationship between the jump length and the run-up speed. Actually, the actors were asked to adapt the most natural run-up speed to execute a jump of a given length. This limits high dynamic variations between the run-up phase and the jump. Therefore, the speed s_2 measured just before the jump execution can be approximated by the mean speed of this jump.

We extract from all jumps in the database their mean speed. This latter is computed by dividing the traveled distance of the character's root during the jump sequence (from RHS to LTO event) by the jump duration (see Eq. 1.10, Part III). Fig. 3.5 and 3.6 illustrate the resulting relation between the jump length l and the mean speed s for walking respectively running jumps. Clearly, the best approximation function of these data is obtained by performing a linear least square. Two functions are then constructed, one for each type of jump, returning the mean speed which corresponds to the run-up speed s_2 of a given jump length:

$$f_W(l) = s_2 = 0.97l - 0.04 \pm 0.1 \tag{3.1}$$

$$f_R(l) = s_2 = 1.16l - 0.13 \pm 0.15 \tag{3.2}$$

Note that we add a tolerance to these functions. Actually, depending on the performer skill level, a given jump length matches several acceptable mean speeds. This speed range is illustrated in Fig. 3.5 and 3.6 by the stripe built from the positive to negative tolerances of the linear approximation.

Therefore, given a jump length, a range of speed values is determined. If the current locomotion speed s_1 is outside of this range, we propose two motion adaptation scenarios:

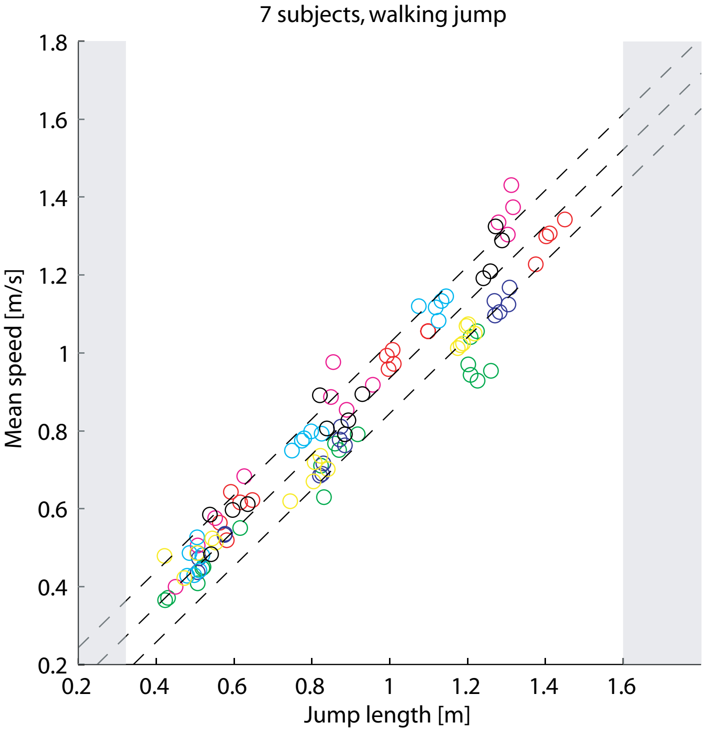


Figure 3.5: Walking jumps: relation between the mean speed and length. Given a jump length, the stripe delimited by the linear approximation (middle dashed line) and the lower and upper tolerances defines allowed speed values just before the jump. The gray zone indicates not commonly feasible jumps.

- The priority is given to the jump execution. A new jump is generated, with the appropriate length and type which correspond to the current locomotion parameter. This solution is well adapted to game environment where the user's actions need to be quickly executed.

- The priority is given to the jump parameters. The current motion is adapted by a continuous speed variation until reaching an acceptable speed the jump requires. Additionally and simultaneously, the current locomotion weight w_{loco} is adapted to the one of the jump. When the jump is finished, the locomotion parameters are gradually restored to their initial values. This solution is interesting when a character has to clear accurately an obstacle having specific dimension.

3.3.1.2 Run-up Duration

The determination of the run-up phase duration t_{runup} is necessary for the second priority scenario, ensuring a coherent transition. We describe a first approach which is based on a constant speed variation.

According to the linear speed variation $\Delta s = s_2 - s_1$, the duration t_{runup} have to be chosen so as the induced acceleration is less than a threshold ε_a:

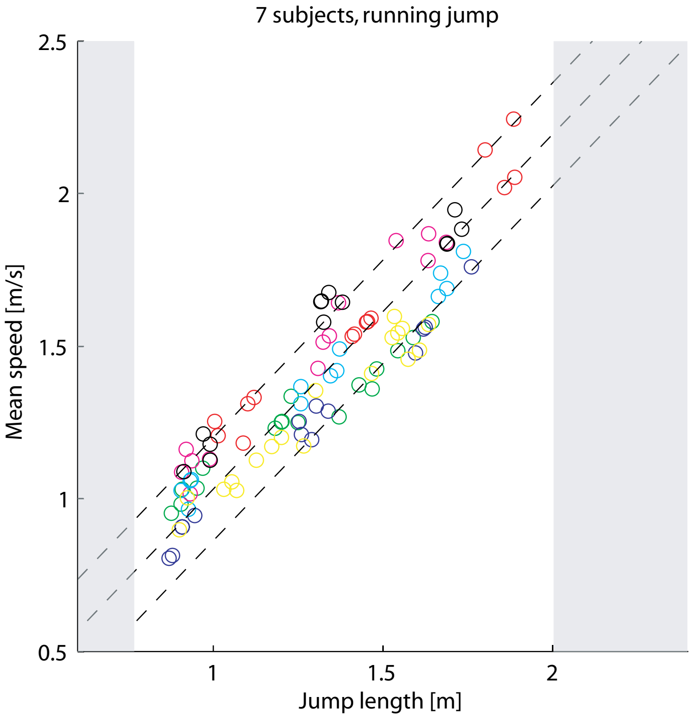


Figure 3.6: Running jumps: relation between the mean speed and length. Given a jump length, the stripe delimited by the linear approximation (middle dashed line) and the lower and upper tolerances defines allowed speed values just before the jump. The gray zone indicates not commonly feasible jumps.

$$a = \frac{\Delta s}{t_{runup}} \leq \varepsilon_a \tag{3.3}$$

To compute t_{runup}, we consider the next possible time when the transition to jump is possible. In fact, a transition starts only at a specific foot event which occurs once during a locomotion cycle, namely when the current phase $\varphi_i = 0 \pmod 1$ (see Sub-section 3.3.2). We define φ_1 the phase at time t_1 and φ_2 the phase at time t_2. The run-up duration can be therefore approximated using Eq. 1.2 (Part III):

$$t_{runup} = t_2 - t_1 = (\varphi_2 - \varphi_1)\frac{2}{F_{loco}(\mathbf{\Psi}_1) + F_{loco}(\mathbf{\Psi}_2)} \tag{3.4}$$

where the frequency variation is approximated by considering it as constant between t_1 (where motion parameters are $\mathbf{\Psi}_1$) and t_2 (where motion parameters are $\mathbf{\Psi}_2$).

In order to determine the number of locomotion cycle to wait before performing the jump, Eq. 3.4 is evaluated by setting $\varphi_2 = 1$. This operation is repeated by adding 1 to φ_2 (i.e. an additional cycle) as long as the returned t_{runup} does not satisfy the acceleration condition described in Eq. 3.3.

However, this approach involves a linear speed variation during this run-up phase and can generate several locomotion cycles before executing the jump. Hence the take-off foot position is not fixed and depends on the current motion and jump parameters. We address this problem later in this thesis (see Part IV, Chapter 1).

3.3.2 Transition Time and Duration

Many works [Rose et al., 1998; Kovar and Gleicher, 2003; Menardais et al., 2004] have demonstrated the necessity to take into account the support phases when blending is performed. In fact, incompatible foot constraints can not be blended (e.g. a right hoping with a left hoping) as they violate the motion properties. These support phases also allows to blend several motions in a synchronous way. The algebraic relation proposed in [Menardais et al., 2004] determines dynamically the transition time by checking the support phases' compatibility of the blended motions. For example, a double support is always compatible with a single support.

However, the double support defined from LHS to RTO in a walking motion (Fig. 3.2 and Fig. 3.4) is incompatible with the single support from RHS to RTO for a jump starting with the right take-off foot. Actually, the right leg is in front of the character trunk at the double support and back at the single support. Our method avoids this drawback by guaranteeing that the blending from locomotion to jump occurs during the single support phase defined between RHS and RTO, and similarly from jump to locomotion between LHS and LTO. In that way, those support phases determine not only the transition time, but also the transition duration.

Still, it remains to detect the foot events from the blended motions. As our method is foreseen to works dynamically, without knowing those motions in advance, these events have to be detected in real-time. For locomotion patterns, we apply the footplant detection method presented in the previous chapter (Section 2.3), which is able to return the foot event's frame indices in the normalized time, at any time and for any motion parameters.

On the contrary to locomotion, it is however difficult to apply this method to jumping motions. The δ value described in Eq. 2.4 of the previous chapter has to be adapted regarding the type and the performer of the jumps. Our experiments show that for various walking jumps performed by different subjects the δ value varies significantly. Fig. 3.7 illustrates the trajectory curves (sagittal plane) and the detected RTO events (circles) of the right heel for different jump lengths performed by two subjects. By applying a $\delta = 0.6$, the RTO events from the subject $v = 1$ are correctly detected. Conversely, those of subject $v = 2$ occur too late compared to the manually labeled events, represented by squares in Fig 3.7, right. A correct detection is obtained with $\delta = 0.3$.

Another example in Fig. 3.8 illustrates the difference of δ according to the jump type performed by a same subject ($v = 7$ in our situation). As the RTO events for walking jumps are correctly detected with $\delta = 0.75$, those for running occur too late compared to the manually labeled events, represented by squares in Fig 3.8, right. A detection matching those squares is obtained with $\delta = 0.35$. One can observe that, for the given RTO event, there is at most only one frame between the badly and correctly detected frame index. However, for high dynamic motion such as running speed, the joint's vertical position of two consecutive frames can differ in more than 10 cm. It is therefore important to label the different foot events correctly.

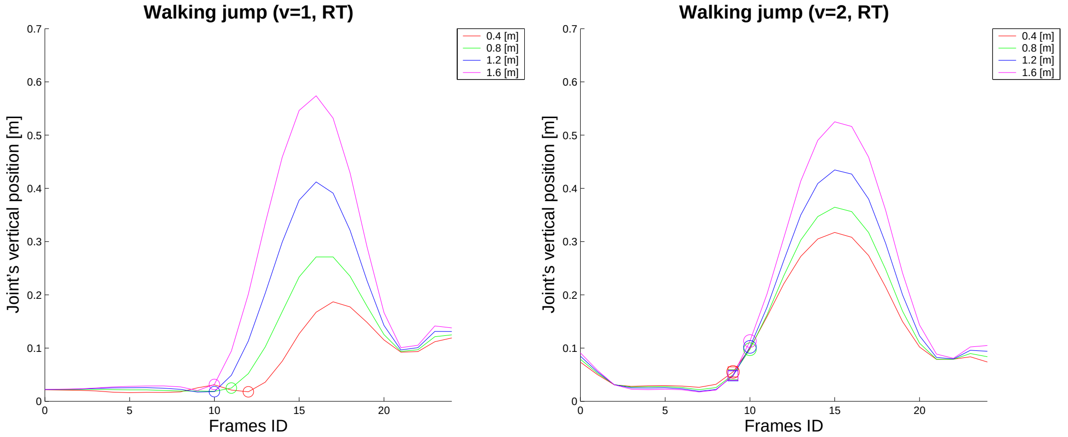


Figure 3.7: The RTO detection result (circles) on jumps for two subjects with $\delta = 0.6$. The curves represent the right heel trajectories projected into the sagittal plane. **Left:** The RHO events are correctly detected (circles). **Right:** The RHO are correctly detected with $\delta = 0.3$ (squares) while the circles indicate the detection results with $\delta = 0.6$.

It becomes evident that for jumping motions, the determination of the δ value is dependent on two parameters: the performer and jump type. For each of those parameter combinations, we need therefore the correct foot event of a selected subset ($\sim 25\%$ of the total jump motion database) of different jump length to choose δ accurately. Hence a subset of jumping motions from each subject and jump type has been manually labeled, as shown in Fig. 3.9. The observation of those foot events leads to the conclusion that given a subject, they are approximately linearly dependent on the jump length. Instead of determining δ values, linear approximations are computed for each foot event, with respect to the subject and jump type.

To summarize, the feet events of the motions generated in real-time can be instantaneously computed. For walking and running, we apply the detection method presented in Part III, Section 2.3, for jumping the method based on linear approximation of manually labeled subset of motions.

3.3.3 Time-Aligned Blending

The support phases of two blended motions may have different durations, depending on the motion properties. For a similar jump, one subject can put the take-off foot longer on the floor than another one. Hence, it is necessary to align in time the support phases which are blended.

Let $\mathbf{A}$ be a motion where frames $\mathcal{F}_{a1}$ and $\mathcal{F}_{a2}$ delimit a support phase which has to be blended with the motion $\mathbf{B}$ having a compatible support phase from frames $\mathcal{F}_{b1}$ to $\mathcal{F}_{b2}$. According to the elapsed time step Δt, we have to compute the time aligned frames $\mathcal{F}_a$ and $\mathcal{F}_b$, as illustrated in Figure 3.10.

While the motions are generated in a generic time and have a same total frame number of N_{frame}, the frequencies F_a and F_b (computed as described in Part III, Section 1.3 in Chapter 2, Eq. 1.8 and Eq. 1.13) associated to the motion $\mathbf{A}$ respectively $\mathbf{B}$ are used to update the current motion phase φ_{cur}. During the blending, we define the frequency F which varies

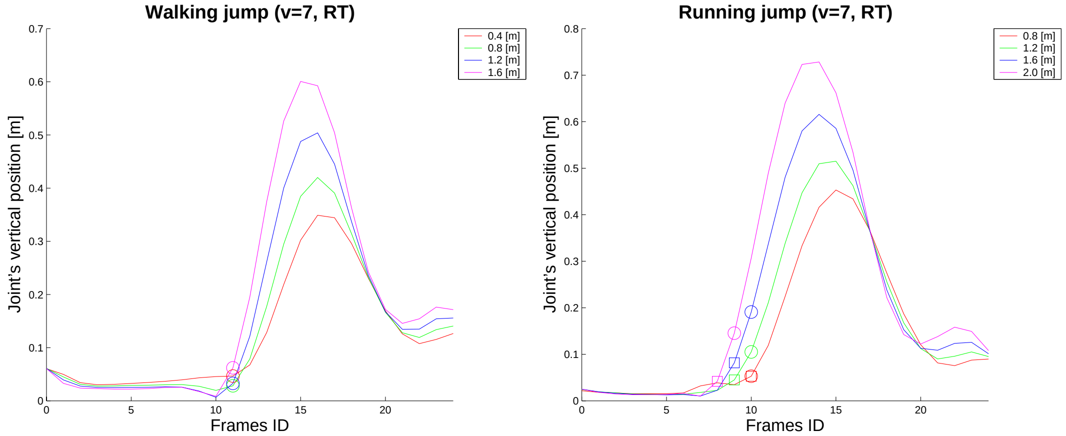


Figure 3.8: The RTO detection (circles) on walking and running jumps for one subject with $\delta = 0.75$. The curves represent the right heel trajectories projected into the sagittal plane. **Left:** The RHO events are correctly detected (circles). **Right:** The RHO are correctly detected with $\delta = 0.35$ (squares).

from F_a to F_b, and a phase variable φ_{cur}. For each elapsed time Δt, the phase is updated as follows:

$$\varphi_{cur} = \varphi_{prev} + \Delta t F \tag{3.5}$$

Hence, the current frame $\mathcal{F}_a$ of the motion $\mathbf{A}$ is computed as described in the next equation (Eq. 3.6).

$$\mathcal{F}_a = \varphi_{cur} N_{frame} \tag{3.6}$$

To find the corresponding time-aligned frame $\mathcal{F}_b$ in the motion $\mathbf{B}$, we define a blending weight parameter p which varies from 0 at the blending start, to 1 at the blending end. This parameter indicates the normalized progression of frames in both motions and is computed as follows:

$$p = \frac{\mathcal{F}_a - \mathcal{F}_{a1}}{\mathcal{F}_{a2} - \mathcal{F}_{a1}} = \frac{\mathcal{F}_b - \mathcal{F}_{b1}}{\mathcal{F}_{b2} - \mathcal{F}_{b1}} \tag{3.7}$$

From this latter equation, $\mathcal{F}_b$ is determined and then interpolated with $\mathcal{F}_a$ to compute the blended frame $\mathcal{F}$. As the frames hold a posture expressed with quaternions, each joint's rotation of $\mathcal{F}$ are computed by the spherical linear interpolation method [Shoemake, 1985] using the parameter p. The root global position is similarly computed by a linear interpolation between the jump's and the locomotion's root position. For this latter, as the patterns are originally generated on a treadmill, we add the displacement corresponding to the current speed. Additionally, the frequency F is computed by performing a linear interpolation from the frequencies F_a to F_b, parameterized with p.

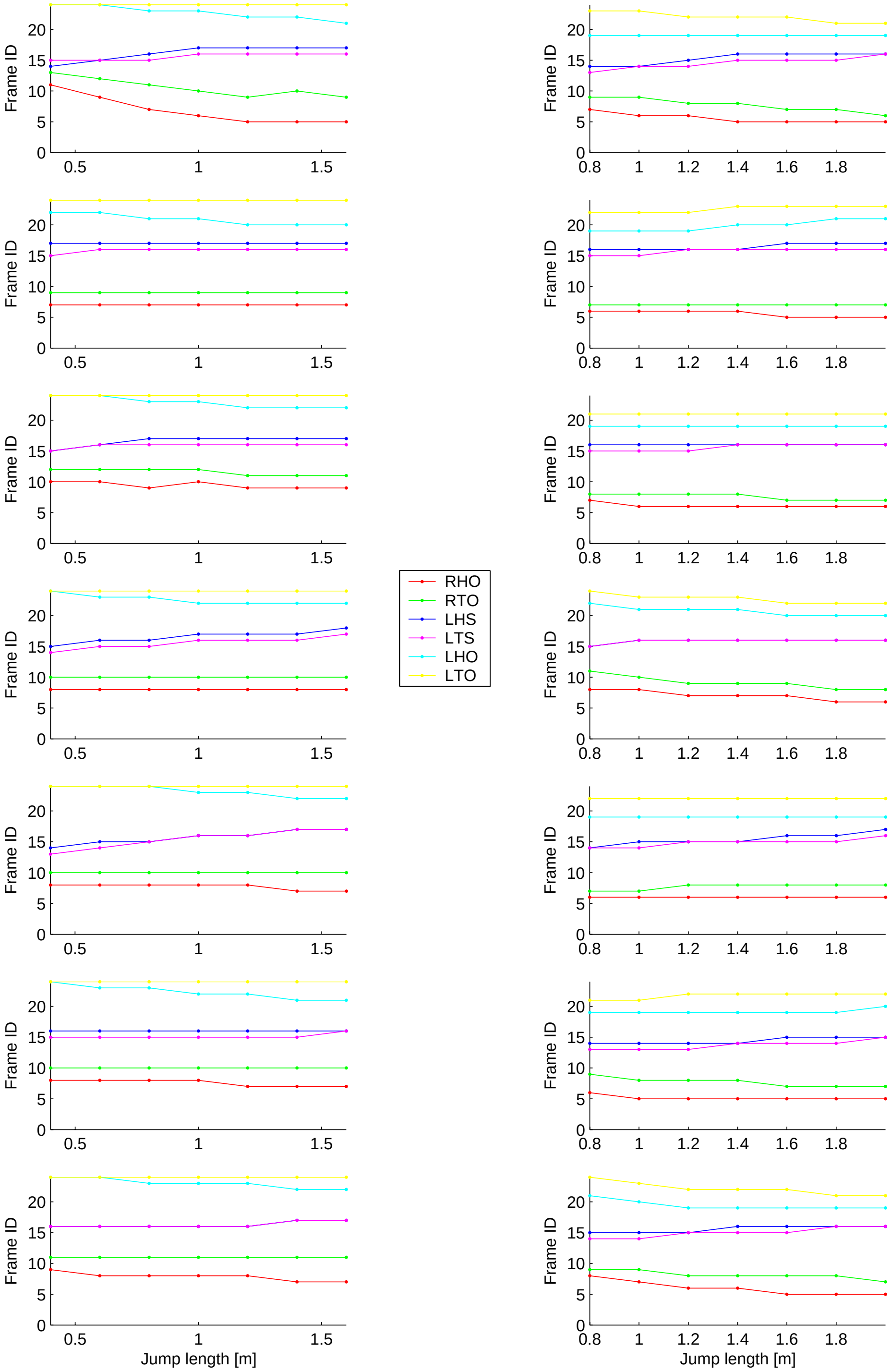


Figure 3.9: Manual labeling of the foot events, for the seven captured subject performing various jump lengths. **Left:** Walking jumps with a length variation from 0.4 to 1.6 [m]. **Right:** Running jumps with a length variation from 0.8 to 2.0 [m].

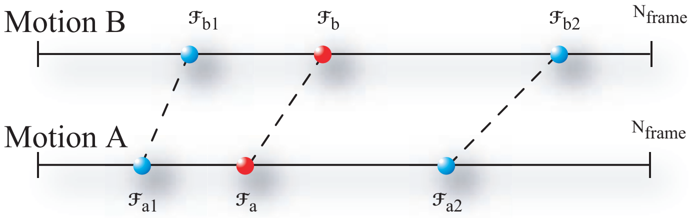


Figure 3.10: Frame alignment between motion A and B

3.4 Results and Conclusion

We have integrated our state machine model in our animation engine. Animators can steer a virtual human by changing the speed s, locomotion weight w_{loco} and the personification $\mathbf{w}_{subj}$. In addition, jumps can be parameterized with a desired length l, personification and type.

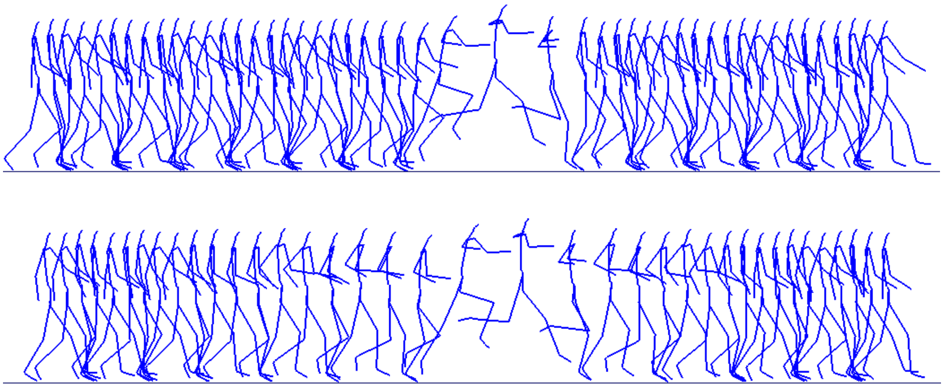


Figure 3.11: Comparison of blending from a walking to a running jump. **Top:** The sequence is generated using traditional blending technique. **Bottom:** The sequence is generated by first modifying the speed and locomotion type before performing the jump. Finally, the motion parameters are smoothly restored to the initial ones.

At any time, the user can trigger a jump. The model adapts automatically on the fly the current motion to preserve speed coherence based on observations made on 105 walking and 103 running captured jumps. Therefore, the speed and type of locomotion may be modified during a specific period of time (Fig. 3.11, bottom). After some experiments, we choose an ε_a value of 0.2 $[m/s^{-2}]$. This value corresponds to the half mean acceleration of a sportsman who sprints a 100 meters race.

Additionally, the method decides when and how long to perform the blending operation to jump, by taking into account the motion support phases. The generated transition is seamless and performed in real-time, having a duration time varying from 0.2 to 0.5 second. Our blending algorithm is based on spherical linear interpolation between the quaternions of each joint, using a linear weight function. Other weight variation schemes such as a cubic step function [Kovar et al., 2002a; Kovar and Gleicher, 2003; Rose et al., 1998] produce similar results with only subtle differences.

Compared to earlier approaches [Kovar et al., 2002a; Arikan and Forsyth, 2002; Lee et al., 2002] on finding possible transition time over large motion capture database, our method presents advantages. First, we do not need to perform a pre-processing step involving all the clips of the database. This allows us to generate dynamically parameterized motions. Then our methodology determines transition time and duration automatically, in contrast to [Kovar et al., 2002a] where transition thresholds have to be specified by hand. Finally, we adapt the current motion by modifying its parameters to enforce the requested blending, unlike previous methods that either do not allow the transition or perform it without adaptation [Kovar and Gleicher, 2003] or at possibly incorrect support phases like in [Menardais et al., 2004].

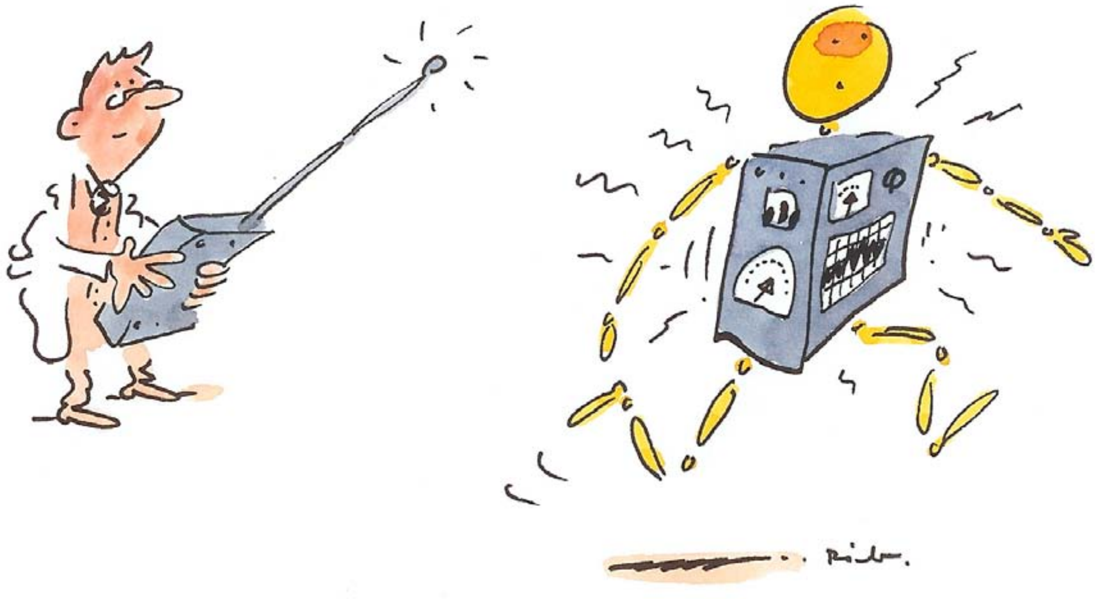
We illustrate the improvement provided by our blending method. The top image of Fig. 3.11 shows a transition from walking to running jump, using a traditional blending method such as [Park et al., 2002a]. One can observe that the jump length is large regarding the step length, leading to a dynamic incoherence. The bottom image shows the same situation by applying our technique. The motion is adapted by modifying the speed and the type of the locomotion according to the requested jump.

Some artifacts may nevertheless occur due to the original motion quality. Therefore, like many other blending methods, we apply a correction algorithm. We use the footplant enforcement method presented in Part III, Section 2.4. This allows to control the foot position, preventing sliding effects. This correction is efficient enough not to alter the real-time performance of the animation.

The motion adaptation phase is performed by a linear speed variation over a specific number of steps, determined according an acceleration tolerance. Consequently, the global position of the take-off foot is variable, according to the motion parameter configuration at the jump's trigger time. However, in some situations, the take-off foot position is important and has to be precisely placed in front of obstacles in order to jump over it without touching it. Hence, in the next part of this thesis, its first chapter introduces a method which determines a speed variation according to the constraint on the take-off foot position.

Part IV

Applications



Chapter 1

Obstacle Handling in Dynamic Environments

1.1 Introduction

In the previous chapters, we essentially address the problem of generating animations with the possibility of varying high-level parameters. These variations can be controlled by an animator, who scripts a scenario by describing the virtual human actions. For example, an action may look like ”From the position P, human H accelerates until reaching the speed S”. While the final animation is generated on-line, the parameter variations are predefined, in contrast to video games. In that case, a user controls the parameter variations of its character in real-time, by means of an interface such a joystick.

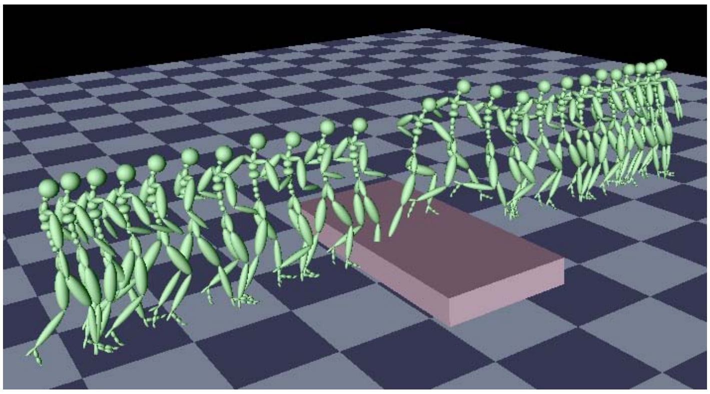


Figure 1.1: A jump sequence without the take-off foot position control. The run-up phase is performed independently of the obstacle position, inducing a collision between the character and the obstacle.

In this chapter, our goal is to apply the animation methods proposed in the previous chapter of this thesis to animate an autonomous character confronted with obstacles similar to the one shown on Fig. 1.1. To represent situations as close as possible to the reality, obstacles are created dynamically, on-the-fly, i.e. at any time during runtime. This scenario simulates the moment when a real human notices an obstacle in front of him.

We have therefore to elaborate a method which first decides whether the obstacle can be jumped over or got round. According to the retained choice, the motion parameters are then consequently modified. This approach is based on mechanisms allowing the navigation around a virtual environment in a life-like and improvisational manner. We refer this concept to as the term *behavior*, as described by Reynolds [Reynolds, 1999]. The author divides the motion behavior into three layers of hierarchy: strategic planning, path determination and animation.

Our purpose is to apply this model to solve the problem of handling obstacles in a dynamic environment. As opposed to traditional methods that consider the objects (or obstacles) of a scene as static [Choi et al., 2003; Pettré et al., 2003a; Go et al., 2004; Sung et al., 2005] or described with pre-defined trajectories [Hsu et al., 2002; Lau and Kuffner, 2005], we aim at handling obstacles that are not known in advance, but generated on-the-fly, during the animation process.

Consider an obstacle that dynamically ”appears” in front of a walking autonomous character. The first level of the motion behavior model selects an appropriate action strategically, regarding the current motion parameters and the obstacle position. Basically, two actions are possible: either to get round the obstacle or to jump over it. Then, the next level elaborates a path the character has to follow. This path determination step entails linear/angular speed and type of locomotion variations, and the determination of a jump length in case of clearing the obstacle. This information is finally sent to the third hierarchical level, responsible for generating correctly the final animation of the character. The schema on Fig. 1.2 summarizes the actions at each level of the behavior model.

We simplify the problem by neglecting the obstacle height, which is constant and very small. Moreover, jump motions starts only with a right take-off foot. To tackle this problem of dynamic obstacle handling, techniques from the robotics are improved, by adding specific human properties to emphasize the behavior believability. In addition, these techniques have to be enough efficient for real-time reactivity. In the next sections, we detail the methodology of each layer.

1.2 Strategic Planning

The first stage concerns the action selection, by determining a strategic planning method. The probabilistic roadmap technique [Kavraki et al., 1996] needs the number and position of all obstacles present in the scene, and therefore can not be applied to dynamic environment. Moreover, in such context, the path from a starting to a goal position is originally free. Hence, as soon as an obstacle prevents a direct way, our planning method decides either to go round or to jump over this obstacle.

An obstacle (illustrated in 3D in Fig. 1.1) is defined by a flat box (e.g. a puddle of water) with a center position $\mathbf{C}_{obs}$, length l_{obs} and width w_{obs}. Let t_1 be the time when an obstacle is dynamically created in front of a virtual human. This virtual human is described with its current position $\mathbf{p}_r(t_1)$ and normalized heading direction $\mathbf{h}(t_1)$. We assume, for convenience,

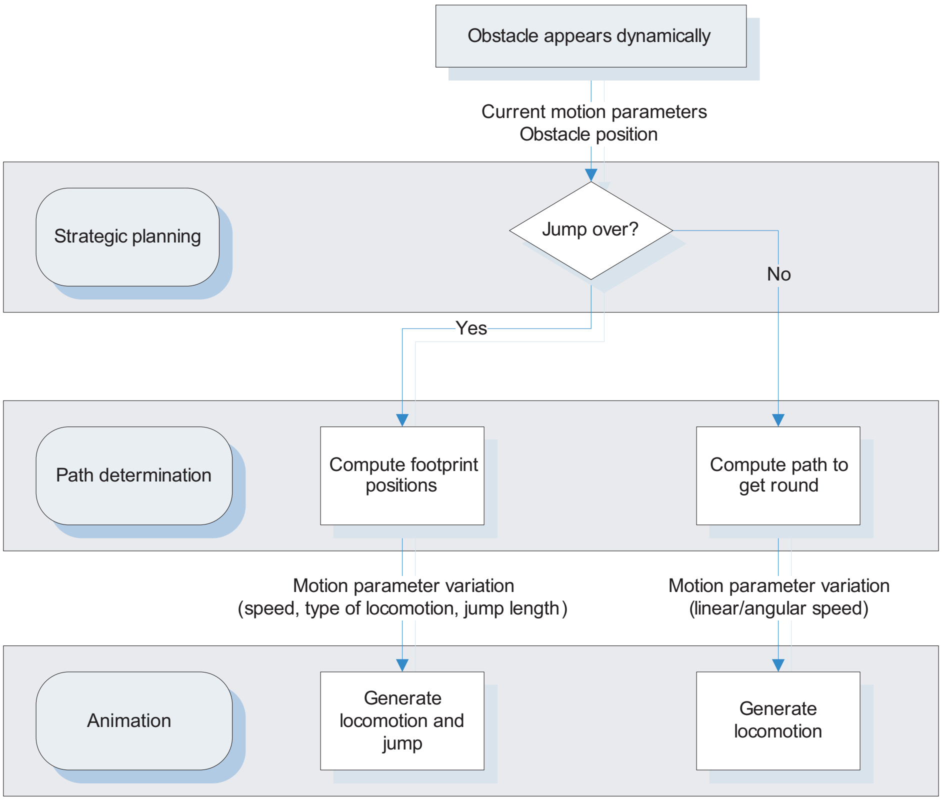

Obstacle appears dynamically
Current motion parameters
Obstacle position
Strategic planning
Jump over?
No
Yes
Path determination
Compute footprint positions
Compute path to get round
Motion parameter variation
(speed, type of locomotion, jump length)
Motion parameter variation
(linear/angular speed)
Animation
Generate locomotion and jump
Generate locomotion

Figure 1.2: The three layers of the motion behavior model for dynamic obstacle handling.

first that this heading direction is constant, and second, that the obstacle faces up a character always perpendicularly and has a constant height. We discuss in the last section of this chapter how this particular case study can be extended to solve more general problems.

The vector from the character position to the front of the obstacle is represented by $\mathbf{d}$. The position just in front of the obstacle is defined as $\mathbf{P}_{start} = \mathbf{p}_r(t_1) + \mathbf{d}$. In addition, at time t_1 the character's motion is parameterized with $\mathbf{\Psi}_1 = (\mathbf{w}_{subj_1}, w_{loco_1}, s_1)$ and angular speed ω_1.

This environment configuration in mind, the choice to clear an obstacle or not is determined by performing different tests. The first one checks if the obstacle length l_{obs} is included in the jump length range that a human is able to performed. Two ranges are defined, from $l_{w_{min}}$ to $l_{w_{max}}$ and from $l_{r_{min}}$ to $l_{r_{max}}$, for walking respectively running jumps. By default, the current locomotion w^1_{loco} determines the type of the jump. However, a requested length between $l_{w_{max}}$ and $l_{r_{max}}$ compels a walking motion to be smoothly modified towards a running one.

Assuming that a given jump length l_j lies between the minimal and maximal boundaries, we compute the required speed s_2 to execute this jump. According to its type and length (i.e the obstacle length w_{obs}), l_j is inserted into either Eq. 3.1 or Eq. 3.2 from Part III, Chapter 3 (dedicated to the transition from locomotion to jump) to determine the corresponding s_2 and w_{loco_2} of $\mathbf{\Psi}_2$, for the time t_2. This time corresponds to the jump start. In the case where $s_2 < s_1$, we avoid decelerating before executing the jump by setting $s_2 := s_1$. The jump

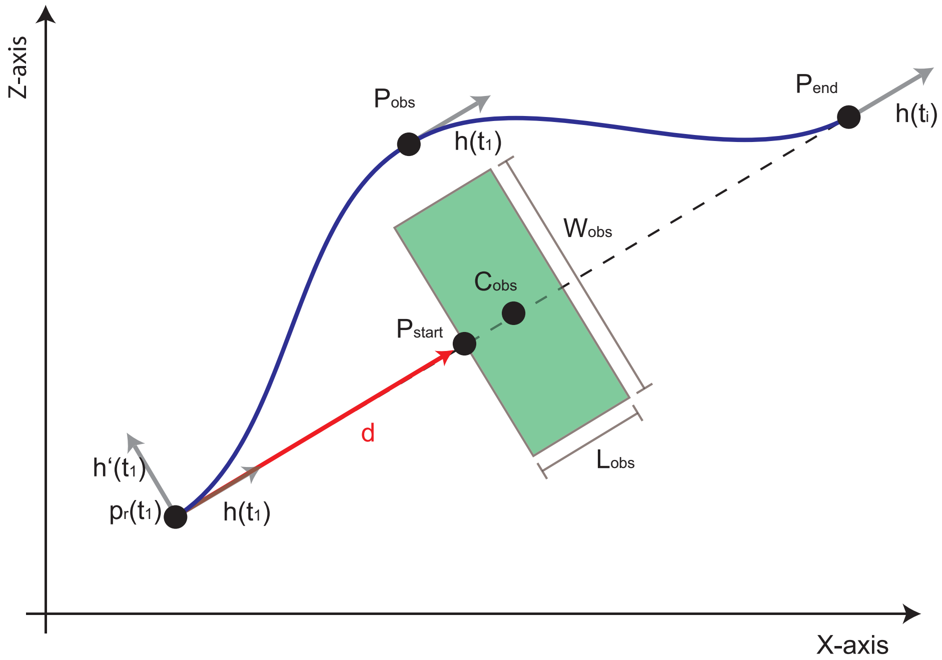


Figure 1.3: Top-view of the environment configuration at time t_1, when an obstacle is generated dynamically. According to the decision planning, either the red or the blue path is determined.

length l_j is therefore adjusted by inserting s_2 into either Eq. 3.1 or Eq. 3.2.

The next test evaluates the mean acceleration $\bar{a}$ computed between the time t_1 and the jump start, as described in Eq. 1.1.

$$\bar{a} = \frac{s_2 - s_1}{\frac{||\mathbf{d}||}{\bar{s}}} \leq \varepsilon_a \tag{1.1}$$

where $\bar{s}$ represents the mean between the s_1 and s_2 speeds. To avoid too abrupt speed modifications, an acceleration threshold ε_a is fixed and corresponds to the maximal acceleration a non sporting human can achieved. Beyond this value, the obstacle has to be got around.

From this first level of our motion behavior model, the presented rules allow an autonomous character to decide either to jump over or to get round the obstacle. The next section focuses on the second behavior level.

1.3 Path Determination

The path determination stage allows the construction of a trajectory which depends on the decision retained at the strategic layer. In the next two sub-sections, we present the methods either to get round or to jump over the obstacle.

1.3.1 Getting Round

When an obstacle can not be jumped over, the character has to get round it by passing either by its right or left side. We arbitrarily choose the left side and we define a point $\mathbf{P}_{obs}$ to go through, depicted in Fig. 1.3 and computed as described in Eq. 1.2, where $\mathbf{h}'(t_1)$ is orthogonal to $\mathbf{h}(t_1)$ (see Fig. 1.3).

$$\mathbf{P}_{obs} = \mathbf{C}_{obs} + \mathbf{h}'(t_1)(\frac{w_{obs}}{2} + \delta l_{obs}) \tag{1.2}$$

The underlying idea is to move this point away from the obstacle regarding its length, in order to avoid a possible collision. We fix therefore a value $\delta \in [0 \ldots 1]$ in order to determine the influence of the obstacle length. In addition, the character's original trajectory is joined by defining a second point $\mathbf{P}_{end}$ to go through, by a symmetry of the point $\mathbf{p}_r(t_1)$ according to the axis traversing the obstacle, perpendicular to $\mathbf{d}$.

The path which avoids the obstacle is then determined by those two points $\mathbf{P}_{obs}$ and $\mathbf{P}_{end}$, with their directions similar to $\mathbf{h}(t_1)$. Bezier curves can be applied to construct a smooth path [Pettré et al., 2003b]. It has then to be transformed into a trajectory respecting velocity and acceleration constraints, a classical problem in Robotics [Lamiraux and Laumond, 1997]. However, we prefer to adopt the steering model proposed in [Boulic, 2005]. This latter allows to go through a desired position (like [Reynolds, 1999]), with a desired heading direction at that position. This steering method ensures the reach of the target, thanks to a process that anticipates either the success or failure of the reach, and adjusts the desired speed accordingly. In addition, pre-calculated data greatly reduces the computational cost, which is appreciable for our real-time context.

Hence, the getting round of the obstacle is performed in two steps. The first one is initialized by setting the target position $\mathbf{P}_{obs}$ and direction $\mathbf{h}(t_1)$ to the steering model. As long as the target is not reached, the model updates at each animation time step the character's linear and angular speed, according to the previous the character's position as well as its linear speed and locomotion phase. As soon as the first target is reached, the second phase is engaged similarly, by setting the new target position $\mathbf{P}_{end}$ and direction $\mathbf{h}(t_1)$.

1.3.2 Jump Run-up Computation

When the jump is feasible, its run-up phase should be generated so as the right take-off foot is finally placed as close to the obstacle as possible and the locomotion speed corresponds to the required jump length. To solve this problem, the algorithm presented in [Choi et al., 2003] may be applied. It consists in generating randomly a series of footprints between the start and the end of the run-up motion, and then in adapting appropriate motion capture clips. Another method which provides less precise results search for a suitable motion capture clip in an enhanced motion graph structure [Lau and Kuffner, 2005]. However, as those methods are interactive and do not ensure the speed constraint, we propose an alternative approach. This allows an on-the-fly generation of a motion sequence which corresponds to the obstacle constraints.

The fundamental idea of our method is to compute a footprint combination matching the following constraint formulation: ”Position the right take-off foot at $\mathbf{P}_{start}$ with a locomotion speed s_2 and type w_{loco_2}. In other words, we have to determine a motion parameter variation $\mathbf{\Psi}(t)$ from $\mathbf{\Psi}_1$ at time t_1, i.e at the obstacle generation, to $\mathbf{\Psi}_2$ at time t_2, when the character is located at $\mathbf{P}_{start}$. First we assume that $\mathbf{w}_{subj_1} = \mathbf{w}_{subj_2} = \mathbf{w}_{subj}$, as these parameters have no implication in the problem of obstacle handling. Second, we define the variation from w_{loco_1} to w_{loco_2} as linear, as validated in a previous chapter (Part III, Chapter 3). Finally, we define the speed variation function (or speed profile) $S(t)$, from s_1 to s_2.

The simpler approach consists in describing the speed variation $S(t)$ with a linear function, involving a constant acceleration, coupled with a constant function, as described in Eq. 1.3. To ensure the character reaches $\mathbf{P}_{start}$ at time t_2, the speed varies from s_1 to s_2 during t_1 and t'. Then the speed is constant until t_2, as illustrated in Fig. 1.4 with different t' values.

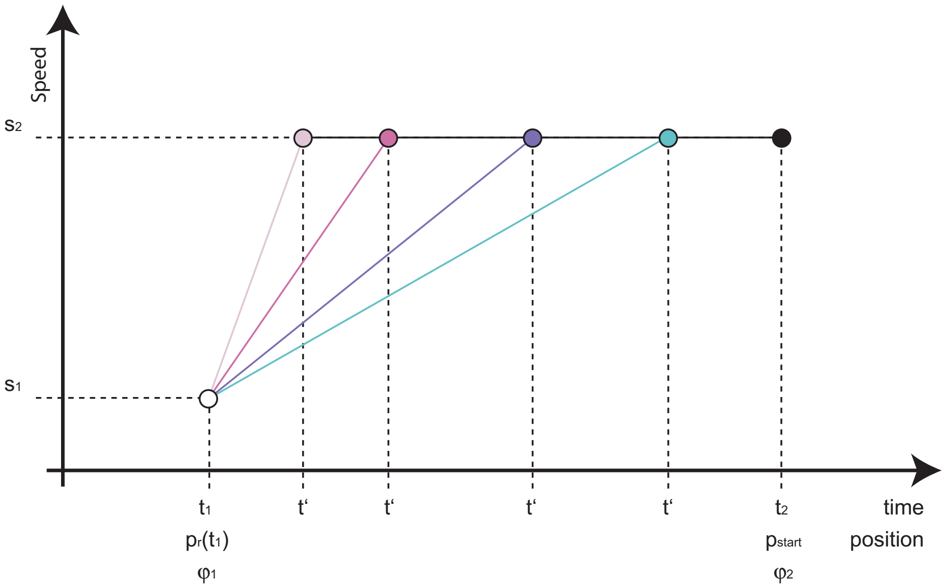


Figure 1.4: Linear speed variation from s_1 at position $\mathbf{p}_r(t_1)$ to s_2 at $\mathbf{P}_{start}$: over different variation scenarios, one have to match the right take-off constraint at position $\mathbf{P}_{start}$.

$$S(t) = \begin{cases} at + b & \text{for } t_1 \leq t \leq t' \\ s_2 & \text{for } t' \leq t \leq t_2. \end{cases} \tag{1.3}$$

However, this approach is not enough generic. For example, if $s_1 = s_2$, the speed can not vary and the phase constraint at t_2 is therefore no more guaranteed. Also if the final desired speed is $s_2 = 0$ (e.g. to stop just in front of an obstacle), t' has to be equal to t_2. Otherwise, the null speed s_2 is set before reaching the position $\mathbf{P}_{start}$.

We propose to tackle those drawbacks by applying another speed variation model, based on a quadratic function:

$$S(t) = at^2 + bt + c \tag{1.4}$$

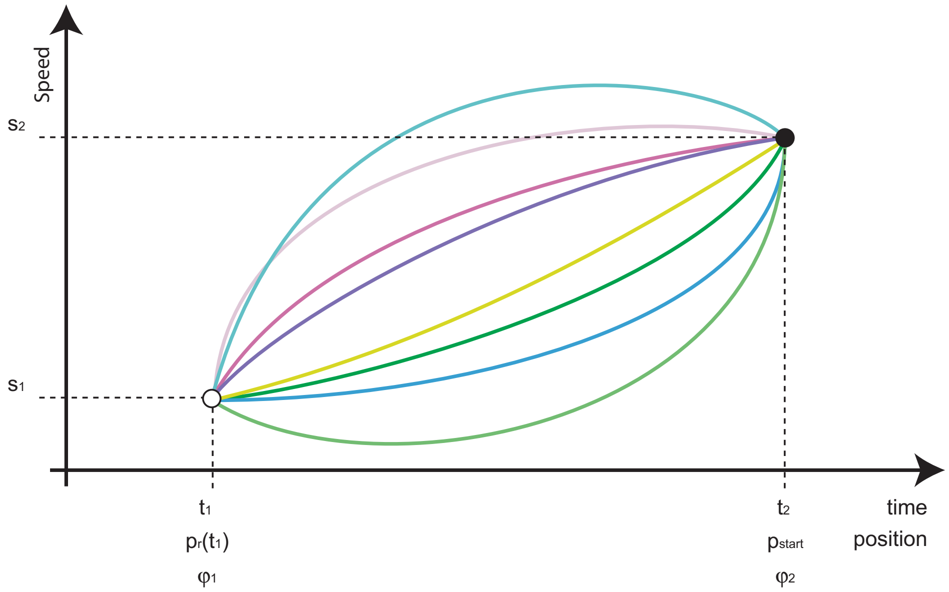


Figure 1.5: Quadratic speed variation from s_1 at position $\mathbf{p}_r(t_1)$ to s_2 at $\mathbf{P}_{start}$. Over different variation scenarios, one have to ensure the right take-off foot at position $\mathbf{P}_{start}$.

The three parameters a, b and c determine the shape of the function in order to match the take-off foot goal position, final speed and locomotion phase conditions (Fig. 1.5). In addition to these unknown, the t_2 duration of the variation is to be computed. In total, four equations have to be determined. The first two match the initial and final conditions of the speed profile, assuming that $t_1 = 0$:

$$\begin{aligned} S(0) &= c = s_1 \\ S(t_2) &= at_2^2 + bt_2 + c = s_2 \end{aligned} \tag{1.5}$$

Then the traveled distance $d = ||\mathbf{d}||$ between $\mathbf{p}_r(t_1)$ and $\mathbf{P}_{start}$ is expressed as a function of time and speed:

$$d = \int_0^{t_2} (at^2 + bt + c)\, dt = \frac{1}{3}at^3 + \frac{1}{2}bt + ct \tag{1.6}$$

Finally, we use the locomotion phase constraint to describe the last equation. Let be φ_1 defined as the current phase of the character at time t_1. In order to ensure the right heel strike event at time t_2, the phase φ_2 has to be an integer value. In addition, this phase corresponds to the cycle number executed between t_1 and t_2. The phase update allows to determiner the last equation, using the frequency function $F_{loco}(\mathbf{\Psi})$ defined in Eq. 1.8 (Part III, section 1.3):

$$\begin{aligned} \varphi_2 &= \varphi_1 + F_{loco}(\mathbf{\Psi})t_2 \\ &= \varphi_1 + \left[\sum_{i=1}^{n} F_{loco}(\mathbf{w}_{subj}, \frac{1}{t_2}((t_2 - \frac{i}{n}t_2)w_{loco_1} + \frac{i}{n}t_2 w_{loco_1}), S(\frac{i}{n}t_2))\frac{t_2}{n}\right] \\ &= \varphi_1 + \int_0^{t_2} F(\mathbf{w}_{subj}, (1-t)w_{loco_1} + tw_{loco_2}, S(t))\, dt \end{aligned} \tag{1.7}$$

However, the solution of the integral in Eq. 1.7 is complex to compute. In fact, the frequency function F_{loco} is defined by two different functions, according to the speed value (see Part III, Sub-section 1.3.1). Therefore, the integral in Eq. 1.7 has to be divided into parts during which the frequency function is either defined by the first or the second function. The number and the duration of these parts are unknown. It introduces an additional difficulty which we aim to avoid to ensure real-time performances.

To solve this problem, the frequency function is expressed as a linear interpolation between the two frequency functions $F_{loco}(\mathbf{\Psi}_1)$ and $F_{loco}(\mathbf{\Psi}_2)$. Hence, we approximate Eq. 1.7 with Eq. 1.8 by assuming that the frequency function is linear between t_1 and t_2.

$$\int_0^{t_2} F(\mathbf{w}_{subj}, (1-t)w_{loco_1} + tw_{loco_2}, S(t))\, dt \approx \frac{F_{loco}(\mathbf{\Psi}_1) + F_{loco}(\mathbf{\Psi}_2)}{2} t_2 \tag{1.8}$$

Finally, the four unknown a, b, c and t_2 are computed by solving the system composed of the four described equations (Eq. 1.5, Eq. 1.6 and Eq. 1.8).

However, the value of φ_2 which actually corresponds to the performed cycle number before the jump, remains to be determined. For given parameters s_1, s_2, φ_1 and d, a too big φ_2 involves a negative speed during a time interval (Fig. 1.6, left). In practice, it means that the number of cycle is so important that the character has to slow down, and even to walk backwards when the speed is negative, which is not the intention of the method. Conversely, a too small φ_2 (i.e. a small number of cycles) entails an abrupt speed variation not physically feasible by the character (Fig. 1.6, right). Fig. 1.7 illustrates the speed variation $S(t)$ (blue surface) for a continuous φ_2 variation. Observe the negative speed variation for some φ_2 values.

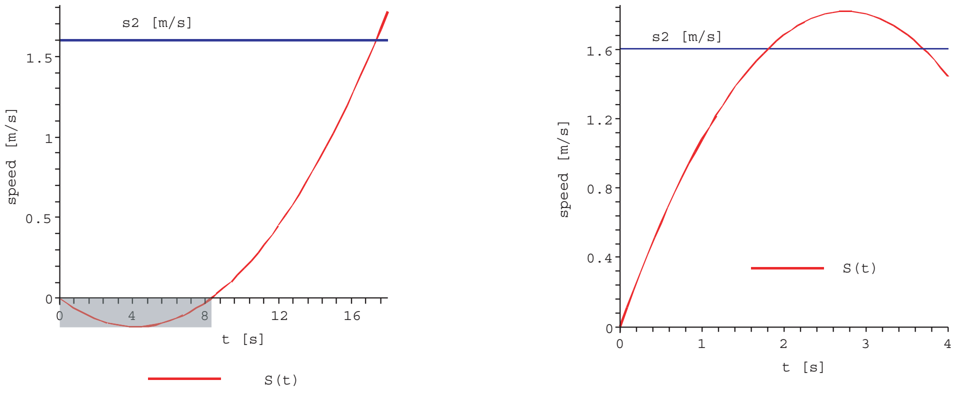


Figure 1.6: Left: φ_2 is too big and violates the condition $S(t) \geq 0$. **Right:** φ_2 is too small and entails a too strong acceleration.

Hence, we determine φ_2 which generates the variation closest to the linear one. Actually this linear variation entails the smallest acceleration variation, which has to be below the acceleration threshold ε_a defined in Eq. 1.1. The algorithm is composed of two phases.

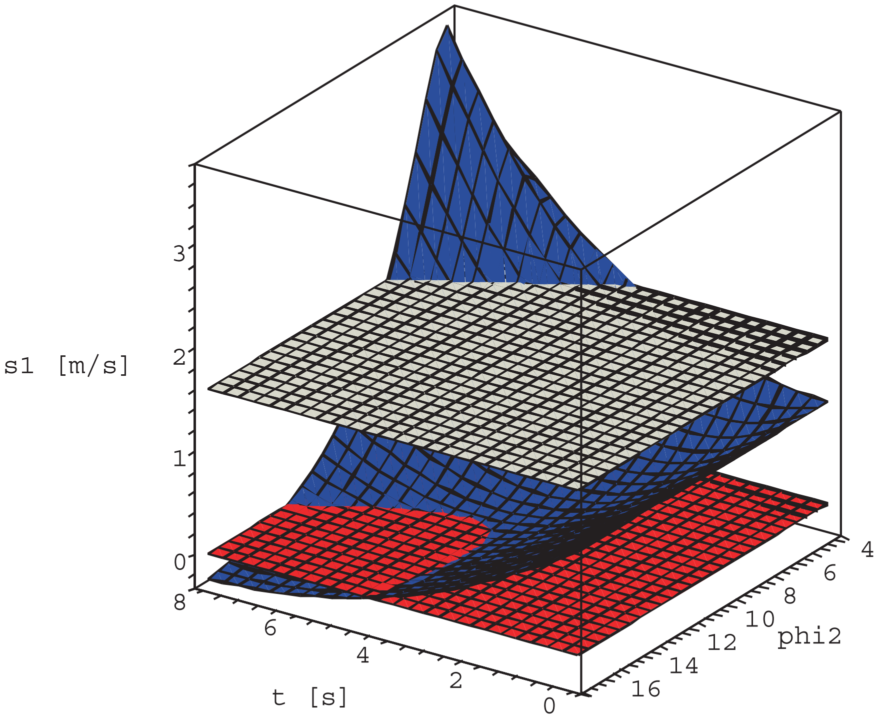


Figure 1.7: The function $S(t)$ (in blue) for different φ_2, given $s_1 = 1$, $s_2 = 1.6$, $\varphi_1 = 0.1$ and $d = 5$. The red plane corresponds to $S(t) = 0$ and helps to visualize when $S(t) < 0$. The white plane represent the final speed ($S(t) = s_2$).

First, φ_2 is initialized by assuming that the speed variation is linear. Using the approximation of Eq. 1.8 in Eq. 1.7 and by computing $t_2 = \frac{2d}{s_1+s_2} = t_{2Lin}$, the final locomotion phase is rounded as follows:

$$\varphi_2 = \left\lfloor \varphi_1 + \frac{F_{loco}(\mathbf{\Psi}_1) + F_{loco}(\mathbf{\Psi}_2)}{s_1 + s_2} d + 0.5 \right\rfloor \tag{1.9}$$

Secondly, this approximated phase value is adjusted iteratively. This value, added to the given input parameters s_1, s_2, φ_1 and d, allow the unknowns a, b, c and t_2 to be computed in order to generate the function $S(t)$. From its derivative $\frac{dS(t)}{dt} = A(t) = 2at + b$, we compute the slope $m = 2a$.

A negative m involves that either the extremum of $S(t)$ is a maximum inside $[0 \ldots t_2]$ or the speed decreases always from s_1 to s_2, as illustrated in Fig 1.8. In both cases, the target phase φ_2 may be increased. Hence, this phase is iteratively incremented by 1. At each i-th iteration, the new parameters a_i, b_i, c_i and t_{2i} are used to compute the acceleration difference Δa_i defined as follows:

$$\Delta a_i = |A(t_{2i}) - A(0)| = |2a_i t_{2i}| \tag{1.10}$$

The algorithm stops when $\Delta a_i \geq \Delta a_{i-1}$ or when the speed variation provides negative speed values, easily tested using the extremum of $S(t)$.

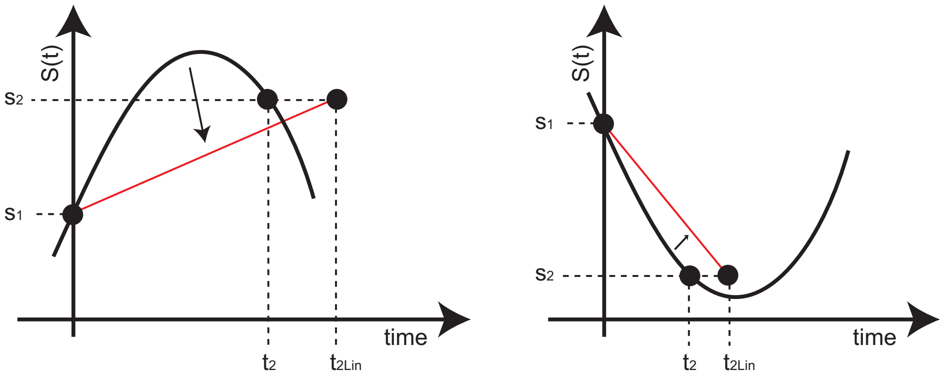


Figure 1.8: Speed variation $S(t)$ (black curve) where the acceleration slope is negative. The arrow indicates that by increasing φ_2, $S(t)$ approaches the red line illustrating the linear speed variation, from $t = 0$ until $t = t_{2Lin}$. **Left:** The maximum is between $t = 0$ and $t = t_2$. **Right:** The extremum is outside $[0 \ldots t_2]$

On the contrary, a positive m indicates that the target φ_2 may be decreased (Fig. 1.9). The algorithm proceeds iteratively by decrementing the phase, and stops similarly to the case where m is negative. In addition, it is ensured that the final $\varphi_2 \geq 1$.

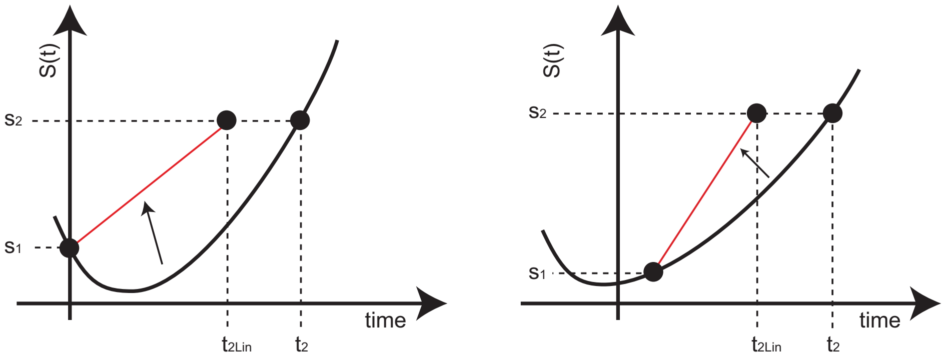


Figure 1.9: Speed variation $S(t)$ (black curve) where the acceleration slope is positive. The arrow indicates that by decreasing φ_2, $S(t)$ approaches the red line illustrates the linear speed variation, from $t = 0$ until $t = t_{2Lin}$. **Left:** The minimum is between $t = 0$ and $t = t_2$. **Right:** The extremum is outside $[0 \ldots t_2]$.

The path determination is therefore completed by providing the motion parameter variations either when getting round or jumping over obstacles. The next section explains how these parameter variations are applied to animate a virtual human.

1.4 Animation

The animation engines proposed in the previous chapters of this thesis are applied to move the character confronted with an obstacle. If the obstacle has to be got around, our locomotion model is continuously controlled only by the new linear and angular speeds provided by the path determination layer. In case of clearing the obstacle, the required jump is first generated. Then the linear speed is modified regarding its variation $S(t)$ computed in the previous layer. The locomotion type modification, if necessary, is performed by a linear interpolation during the speed variation. When $S(t)$ reaches the final speed s_2, we execute the blending from locomotion to jump and inversely by applying the method described in Chapter 3 (Part III). Finally, just after the jump, the current speed and type of locomotion are restored to the initial speed s_1 and locomotion weight w_{loco_1}.

1.5 Results

In our developed real-time application, a jump can be generated at any time by the animator. The jump dimension and its distance to the current character's position are generated randomly over a predefined range of values. The distance is selected between 2 and 10 [m], the obstacle length between 0.4 and 2.6 [m] and width between 1 and 6 [m]. The obstacle height is constant and set to 0.1 [m].

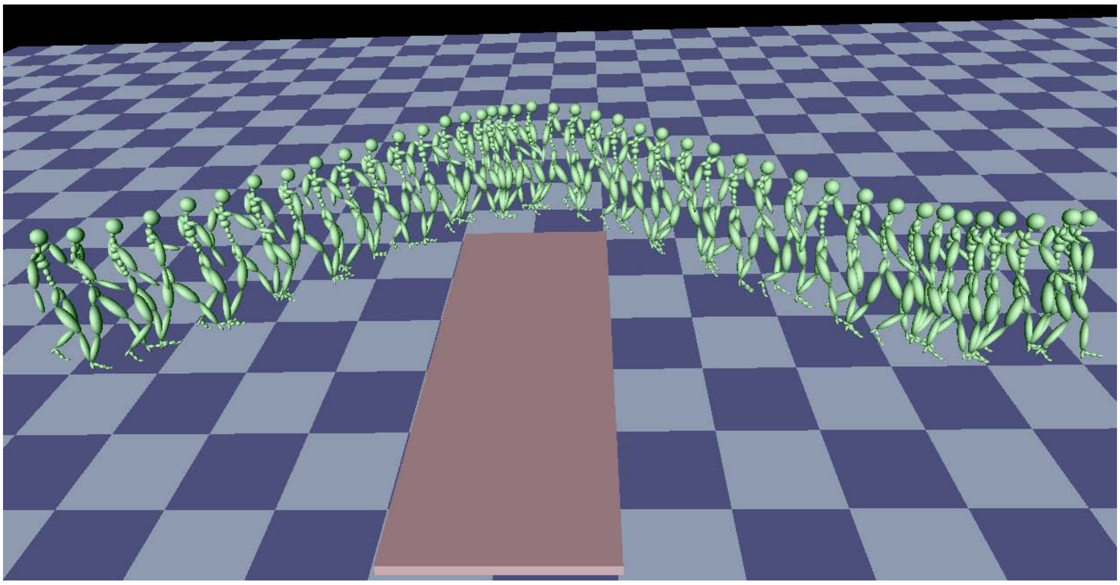


Figure 1.10: The obstacle is got round by the character.

We present the results when the character gets round the obstacle. Fig. 1.10 illustrates the situation where the obstacle can not be jumped over. In fact the final run-up speed necessary for this jump length entails a too important acceleration starting from the initial character locomotion speed. The path goes through the first way-point placed on the obstacle side and the second one placed in order to recover the path followed if the obstacle were jumped over. In the resulting generated animation, we notice that the linear speed is significantly decreased near to the two way-points. This is due to the steering method we used. In fact, this method is highly dependent on a database of pre-computed trajectories. At any time and for a given locomotion parameter situation, the method searches for a correspondence in this database. If it fails, the current parameters are modified in order to find a correspondence. In our case, the speed is automatically decreased to match a trajectory in the database.

In situations where the character jumps over the obstacle, we illustrate two speed variation results, explaining the optimal φ_2 computation. In the first example (Fig. 1.11), the approximated $\varphi_2 = 13$ returns a negative slope m. The phase is augmented until reaching the variation closest to the linear one. As the curve defined with $\varphi_2 = 14$ leads to a Δa greater as the previous one, $\varphi_2 = 13$ is selected. For a pedagogical purpose, the graph on Fig. 1.11 depicts curves from $\varphi_2 = 9$ to $\varphi_2 = 14$.

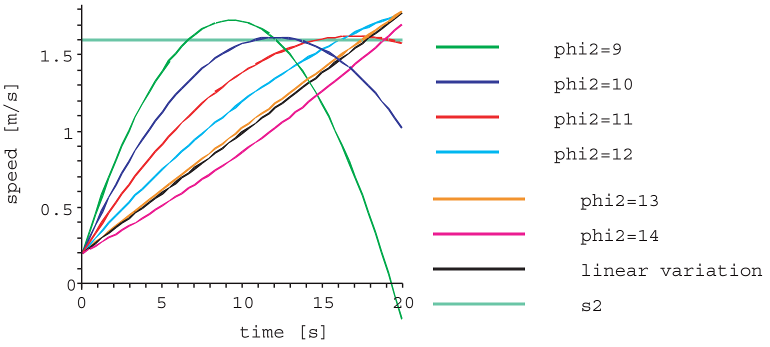


Figure 1.11: Increase of the run-up cycle number. Starting from $\varphi_2 = 13$, $\varphi_2 = 14$ is selected among different speed variations $S(t)$ for a given $\varphi_1 = 0.3$, $s_1 = 0.2$ [m/s], $s_2 = 1.6$ [m/s] and $d = 16$ [m].

The second example (Fig. 1.12) illustrates the situation where the phase φ_2 is iteratively decremented until the optimal value $\varphi_2 = 1$. The corresponding speed variation shows two t_2 solutions to reach s_2. However the solution is unique ($t_2 = 2.09$) due to the φ_2 constraint. One can observe that the variation with $\varphi_2 = 2$ provides in absolute a less accelerated solution. However, our method is based on the relative acceleration difference between the start and end of the variation. This is the reason why $\varphi_2 = 1$ is preferred, as this solution offers less difference, despite that the absolute accelerations at $t = 0$ and $t = t_2$ are high.

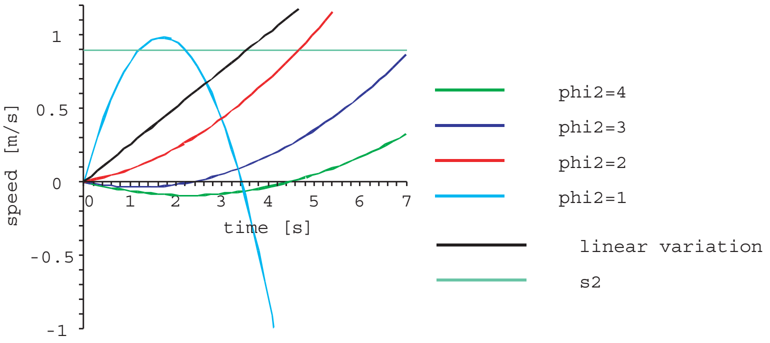


Figure 1.12: Decrease of the run-up cycle number. Starting from $\varphi_2 = 4$, $\varphi_2 = 1$ is selected among different speed variations $S(t)$ for a given $\varphi_1 = 0.1$, $s_1 = 0$ [m/s], $s_2 = 0.9$ [m/s] and $d = 1.6$ [m].

We illustrate in Fig. 1.13 a scenario where the current locomotion speed and type are modified in order to clear the obstacle. According to the obstacle dimension, the performed jump needs a fast final run-up speed, executed by running. Our method computes the footprints positions (i.e. the speed variation) so as to place the right take-off foot as close as possible to the obstacle, with the required speed and type of locomotion. After jumping over the obstacle, the motion parameters are restored to the initial ones.

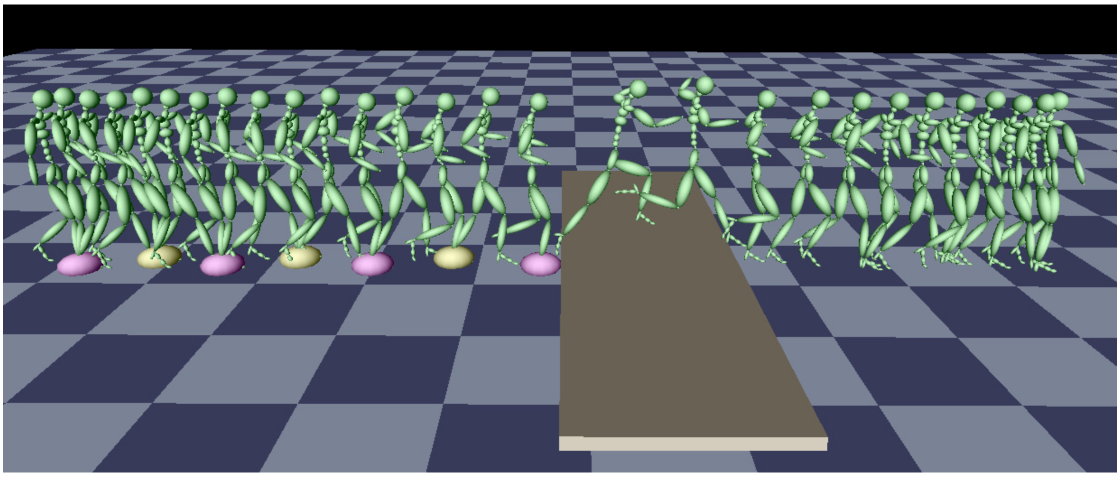


Figure 1.13: The obstacle is jumped over. From a slow walking to a fast running, the method computes the footprints positions (right foot in purple, left in yellow) in order to place the right take-off foot as close as possible to the obstacle.

Our model constraints the current locomotion phase to be null when the 3D root position $\mathbf{p}_r$ is in front of the obstacle. However, the right foot at this phase is forward positioned regarding $\mathbf{p}_r$, as the legs are apart in order to perform the next step. We therefore reduce the distance d by the resulting step length at speed s_2. In that way, the right foot is exactly placed in front of the obstacle.

In practice, we enhance the state machine presented in Chapter 3 (Part III) by embedding the three presented behavioral layers, as illustrated by Fig. 1.14. Thanks to the performance of our method (about 4μ sec on average to compute a speed variation), we are able, in addition to generate the required jump, to compute the speed variation at any time during the animation. It is therefore possible to animate an autonomous character which handles obstacles appearing dynamically.

We have also evaluated the quality of our method. We run our method in a dynamic environment, where obstacles are generated on-the-fly by randomly selecting their dimensions and their position to the moving character. The error E is computed by subtracting from the start obstacle's position $\mathbf{C}_{obs} - \frac{1}{2}\mathbf{L}_{obs}$ the effective position of the right toe $\mathbf{P}_{toe}$ at the jump start. Hence, a quantitative measurement of the method precision is performed. Histograms on Fig. 1.15 and Fig. 1.16 illustrate the experiment results, by counting the jump number with respect to the precision of the take-off foot position. The error is computed for 353 obstacles jumped over with a constant initial walking speed $s_1 = 1.1$ [m/s], respectively for 334 obstacles with a running speed of $s_1 = 1.5$ [m/s]. The foot is correctly placed within an error of 5 [cm] (for a character's leg length of 0.88 [cm]) in more than the 70% of the jumps. The remaining 30% are decreasingly distributed until a maximal error of 20 [cm]. In addition, we try to quantify the influence of the frequency function approximation (Eq. 1.8) on

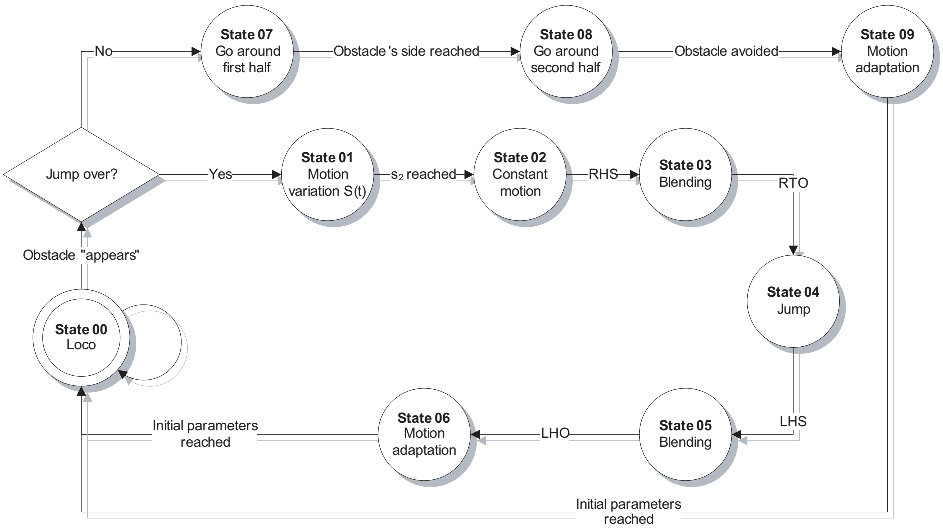


Figure 1.14: The state machine allowing either to go round or to jump over an obstacle.

E regarding speed difference between s_1 and s_2. At a first thought, we can intuitively induce that the lengthier the jump, the bigger the error. The graphs on Fig. 1.15 and Fig. 1.16 depict each jump (represented by a circle) according to its length and take-off foot position error. We conclude that that E is not directly influenced by big length jumps (i.e. final speed s_1).

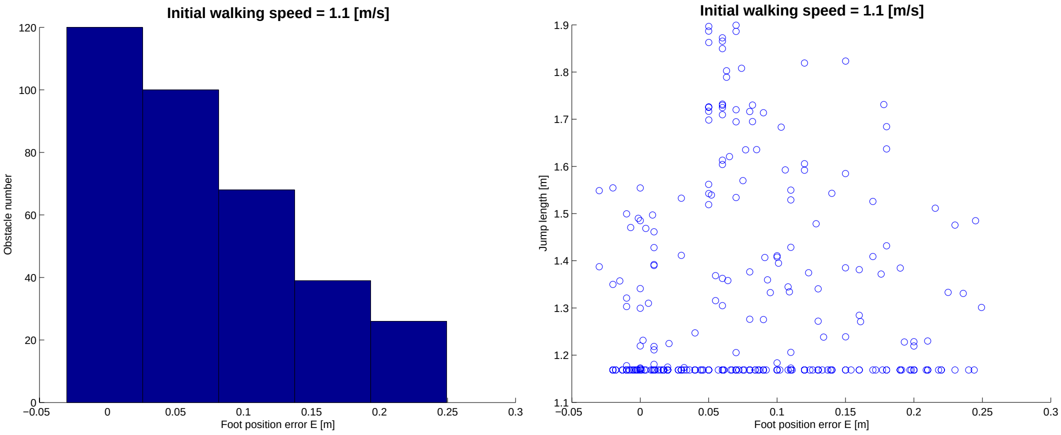
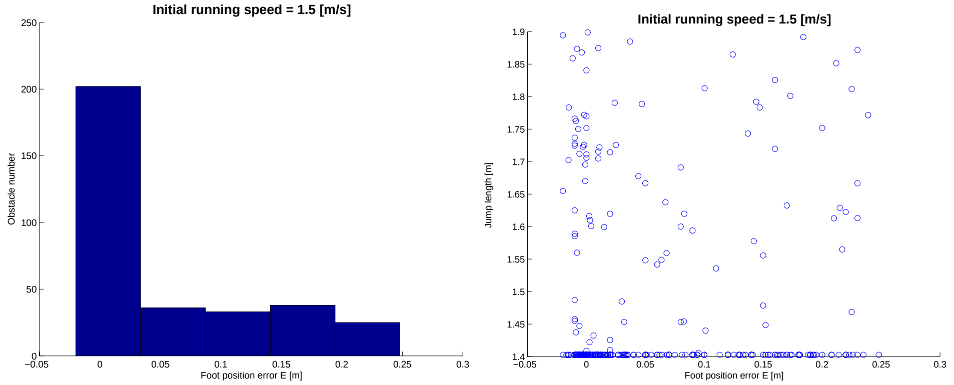


Figure 1.15: Results of 353 obstacles jumped over with an initial walking speed $s_1 = 1.1$ [m/s] **Left:** Histogram representing the measured error E. **Right:** Comparison between the measured error E and the jump length to jump over an obstacle.

1.6 Conclusion and Discussion

In this chapter, we have presented a new behavior model for obstacle handling in dynamic environments. When an obstacle of a specific length suddenly appears in front of a moving autonomous character, the method chooses first whether to go round or jump over the



Initial running speed = 1.5 [m/s]

Initial running speed = 1.5 [m/s]

Figure 1.16: Results of 334 obstacles jumped over with an initial running speed $s_1 = 1.5$ [m/s] **Left:** Histogram representing the measured error E. **Right:** Comparison between the measured error E and the jump length to jump over an obstacle.

obstacle. For the first choice, the linear and angular speeds vary to generate a path which avoids the obstacle. For the second choice, a quadratic speed variation is computed. This latter ensures that the right run-up foot is placed just before the obstacle and that the character has an appropriate speed allowing performing the required jump length. The method is fully dynamic as the motions are generated on-the-fly, without knowing about the environment topology. In addition, the method performance (4μ sec) allows to maintain real-time capability of the animation.

Comparing to similar previous work [Lau and Kuffner, 2005], our method is based on a continuous parameter animation. The motion resulting precision is therefore not dependent on the database size as the exact desired motions are always available thanks to our animation engines (locomotion and jump). Furthermore, our method does not need to know in advance the obstacle location over time. In case of moving obstacles, we can iteratively apply our motion planning method to correct the speed variation.

The steering method used to get round the obstacle decreases the linear speed near to the way-points. This problem can be solved by placing two way-points near to the obstacle side to subdivide the problem complexity. Another idea is to place two way-points to recover the original path, one with the position and the other for the direction.

Our speed variation method is also adapted for other scenario. In fact, it can generate according initial and final conditions set on the locomotion speed and foot positions. Imagine that the character has to jump over a low wall from a standing position (without run-up). The speed variation is computed by setting to zero the speed to be reached at the obstacle beginning. Another interesting problem solved by our method concerns the locomotion combined with stairs climbing. It is important to be sure that either the right or the left foot is placed just in front of the first stair. In this case, the method computes a speed variation ensuring that the final phase equals either 0 (right foot) or 0.5 (left foot). However, this final phase can take any other value according to the desired task.

During the run-up phase, the model estimates the locomotion frequency variation by a linear function. This approximation allows a drastic computational costs reduction without decreasing the result quality. Actually, even when the speed variation is important, the foot is placed in average with less than 10 [cm] close to the obstacle. We can nevertheless ensure that the foot never touches the obstacle by reducing the distance to the obstacle and increasing the length of the jump. The precision of the take-off foot position can be improved by re-computing our speed variation during the run-phase, at a specific sampling rate.

The choice of a quadratic variation model is motivated by its efficiency and flexibility, allowing to handle special cases such as identical initial and final speeds. However, our method induces an acceleration discontinuity, even though we limit the acceleration difference during the variation. Other polynomial variation models of higher degree can be elaborated, in order to add constraint for ensuring acceleration continuity. Still, our opinion is that those models increase the computational costs and do not guarantee a significant improvement in the animation believability. Actually, we think that the human eye is not extremely sensitive to acceleration discontinuities as our results provide already pleasant animations.

Other obstacle types can be handled with our method. For an obstacle which is different from a box and does not face up the character perpendicularly, we construct a bounding box around it. This bounding box, illustrated in Fig. 1.17 (left), defined as rectangular and facing up the character perpendicularly, is then used to solve the jumping problem. Our method can also be adapted for situations where the character walks towards the obstacle with an angular speed. For example when a character turns at a street corner, and sees an obstacle as schematized in Fig. 1.17 (right). As the distance d (from the character position to the obstacle) is function of the angular speed, the strategy consists first in determining a range of fixed angular speed variation. Then, for each of those variations, the distance is computed in order to determine the linear speed variation. It results a beam of trajectories from the character position to the obstacle (Fig. 1.17, right). Finally, from this beam, one trajectory has to be selected.

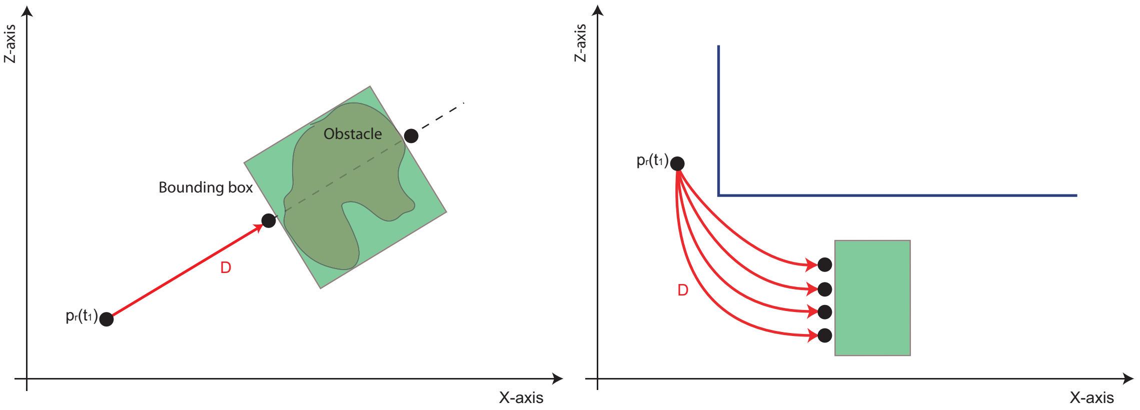


Figure 1.17: Top-view with different obstacle types and situations. **Left:** The obstacle is wrapped by a bounding box. **Right:** Beam of possible character trajectories, with different angular speed variation.

To conclude, we do not consider the obstacle height. However, our method can be easily adapted for such situation. At the strategic planning layer level, the parameter height may lead to other decisions like a different final run-up speed or the stop in front of the obstacle. In this latter case, our method provides a speed variation given the final speed equal to zero.

Chapter 2

Software Implementation

In this chapter, we present the main software tools which have been implemented for our animation engine. It consists of three software modules: the first to construct the hierarchical structure of our motion database, the second in form of two library named *libLOCO* to generate parameterized animations and *libMOCO* to detect and enforce constraints, and finally the third to control and visualize results on virtual humans.

2.1 PCA Hierarchical Structure

The construction of the hierarchical structure composed of PCA spaces is performed by two functions written in Matlab. This process is performed once, in a pre-processing phase.

The first allows to read animation files contained in a directory which compose a motion database. In these files, the representation of the positions and orientations of each joint over time is based on the VRML format. The positions are normalized by the leg length of the captured subject while the rotations are converted into exponential maps. This process is repeated for all motions from the database, and finally it results in a motion matrix where each column represents a motion.

The second function applies a PCA algorithm to this motion matrix and builds the successive hierarchical levels. At each level, the data are separated into groups with a standard K-means algorithm provided by Matlab. PCA is applied to each group and its results are written into a file.

2.2 LibLOCO and LibMOCO Libraries

The animation engine is implemented in library form called *LibLOCO*. Technically this library is written in C++ and has dependencies on existing libraries developed in our research lab. These libraries allow the abstract representation of virtual humans with local and global joint matrices. They also provide functions for matrix algebra computation, and a numerical Inverse Kinematic solver.

Our library allows the creation of an object which represents an instance of the animation engine. During its creation, the files created by the Matlab functions are read according to the hierarchical structure. This latter is stored into the run-time memory. In addition, the skeleton of a virtual character (described using the H-Anim standard) is created and assigned to this engine. Then, the animation parameters are assigned throught an interface of public functions. Finally the motion is generated by updating the skeleton with respect to a given elapsed time.

In connection to this library, another one called *LibMOCO* has been developed to control a motion by detecting and enforcing foot constraints. We decided to write it separately from the animation engine so that this library can be used with other animation tools. *LibMOCO* takes as input the animated skeleton and an instance of an IK solver. As an output, the skeleton is controlled according to the foot constraints.

2.3 Applications

The previous described *libLOCO* and *LibMOCO* are linked to two applications, one to generate animation by scripts and the other to animate a character in an on-line manner.

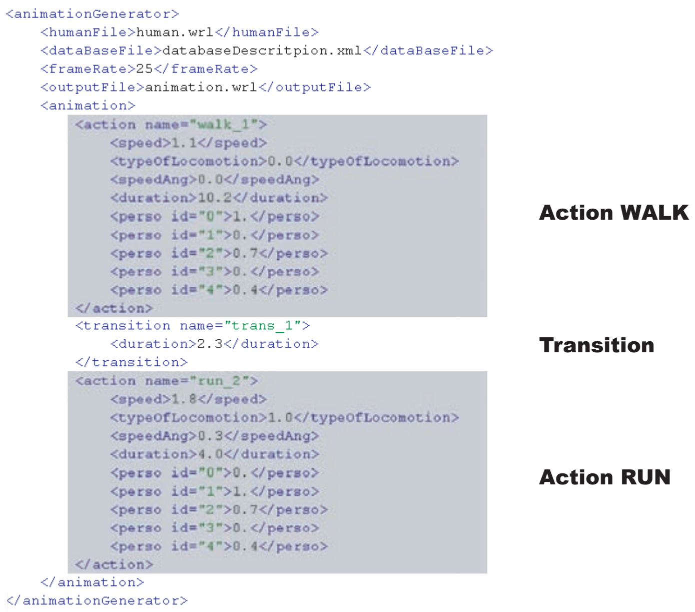
2.3.1 Scripted Motion Generation

The application, called *AnimaGene*, allows to generate an animation by the mean of a XML script. The concept is based on a time-line on which different actions and transition elements are successively placed in a chronological order. Each action and transition element has a defined duration.

An action is either locomotion or jump. For the locomotion, the action is characterized by the parameters personification, walk/run, linear and angular speed. The modification from a parameter configuration to another one is determined by a transition element. The jump action is characterized by the parameters personification, walk/run jump and length. Optionally, the position of the take-off foot can be specified. The transition from the current locomotion parameter to the requested jump is performed automatically and do not need a transition element.

Concretely, an animator writes the list of parameterized actions and transition into an XML file according a specific syntax (see Fig. 2.1 for an example). Then *AnimaGene* scans and parses this file and produces the resulting animation into a destination file. The output format is VRML so as to play the animation in a simple and light 3D viewer.

Note that this application has been used by professional animators in the framework of a European project called MELIES. They generated a small movie where a man was walking along a harbor.



```
<animationGenerator>
    <humanFile>human.wrl</humanFile>
    <dataBaseFile>databaseDescritpion.xml</dataBaseFile>
    <frameRate>25</frameRate>
    <outputFile>animation.wrl</outputFile>
    <animation>
        <action name="walk_1">
            <speed>1.1</speed>
            <typeOfLocomotion>0.0</typeOfLocomotion>
            <speedAng>0.0</speedAng>
            <duration>10.2</duration>
            <perso id="0">1.</perso>
            <perso id="1">0.</perso>
            <perso id="2">0.7</perso>
            <perso id="3">0.</perso>
            <perso id="4">0.4</perso>
        </action>
        <transition name="trans_1">
            <duration>2.3</duration>
        </transition>
        <action name="run_2">
            <speed>1.8</speed>
            <typeOfLocomotion>1.0</typeOfLocomotion>
            <speedAng>0.3</speedAng>
            <duration>4.0</duration>
            <perso id="0">0.</perso>
            <perso id="1">1.</perso>
            <perso id="2">0.7</perso>
            <perso id="3">0.</perso>
            <perso id="4">0.4</perso>
        </action>
    </animation>
</animationGenerator>
```

Action WALK
Transition
Action RUN

Figure 2.1: A script based on the XML syntax used for *AnimaGene*. This example describes an animation sequence composed of a walking followed by a running.

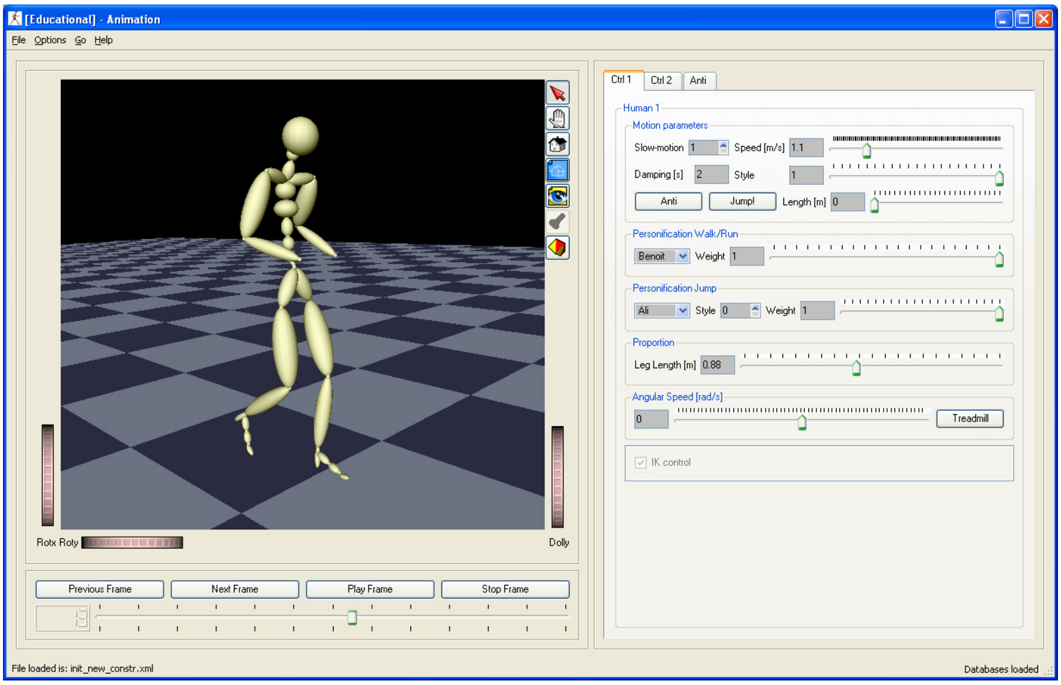
2.3.2 On-line Motion Generation

Another application was developed for the on-line generation of motions. Called *AnimationTool*, this application consists in a graphical user interface (GUI) designed in QT with an embedded 3D viewer based on Open Inventor. This viewer is placed on the left part of the application (see Fig. 2.2). The application displays rigid, non-skinned virtual humans according to the skeleton created in the instances of the animation engine. The proportion of these virtual humans can be modified on-line by changing their leg lengths.

To animate these characters, the interface on the right part of our application (see Fig. 2.2) allows the on-line control of each characters. All parameters can be modified by either inserting a numerical value of by moving sliders. It is also possible to enforce the reach of new parameters values during a given duration.

To make a character jump, the user has to press on a button. In that case, the character motion is automatically adapted before performing the jump in order to be coherent with the required jump. To create obstacles dynamically in the 3D scene, the user has to simply press a control key which activates the random creation of an obstacle in front of the controlled character. The application generates the corresponding animation so as to jump over or get round the obstacle according to the parameter configuration.

One of the main goals of this *AnimationTool* application was to get a simple system for testing our research work. Therefore, the basic design of the characters and the 3D environment was intentional in order to focus on the animation problems. However, as our engine can serve for other applications like Virtual Reality training systems for example. Hence, our application has been integrated into VHD++ development framework [Ponder et al., 2003]. VHD++ is targeted for the rapid component-based development of 3D applications with a fo-



Slow-motion 1
Speed [m/s] 1.1
Damping [s] 2
Style 1
Length [m] 0
Benoit Weight 1
Ali Style 0 Weight 1
Leg Length [m] 0.88
Angular Speed [rad/s] 0
☑ IK control
File loaded is: init_new_constr.xml

Figure 2.2: Interface of *AnimationTool*.

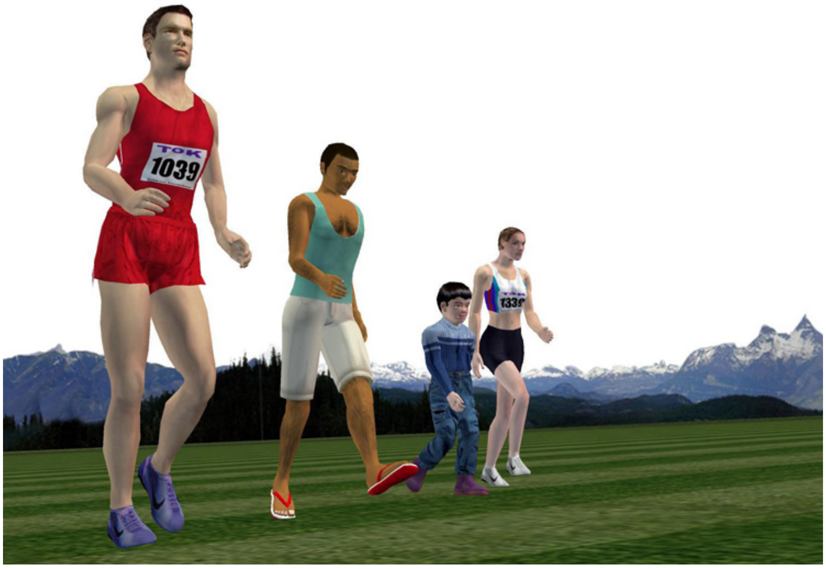


Figure 2.3: Different virtual characters animated by our engine in the VHD++ platform.

cus on virtual humans. It allows among others the rendering of large 3D scenes and skinned characters. Fig. 2.3 illustrates four characters animated by our locomotion engine in the VHD++ platform, while Fig. 2.4 and Fig. 2.5 show different jumps.

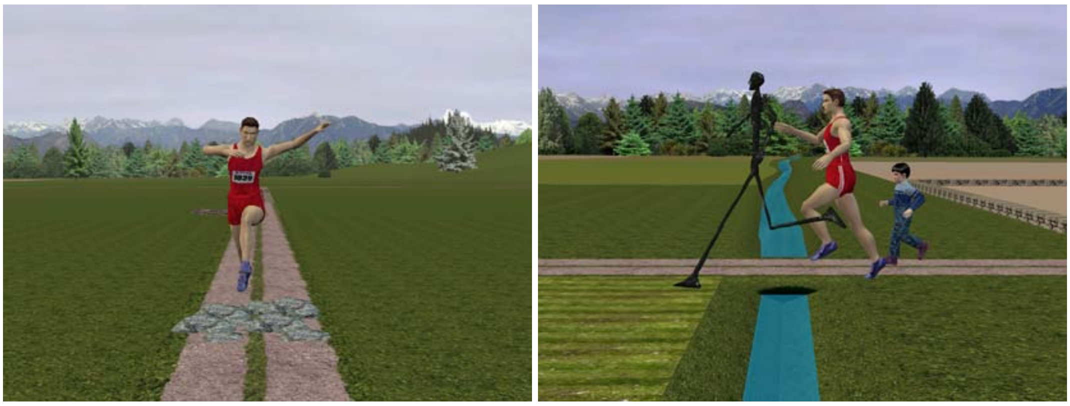


Figure 2.4: Different virtual characters performing various jumps in the VHD++ platform.

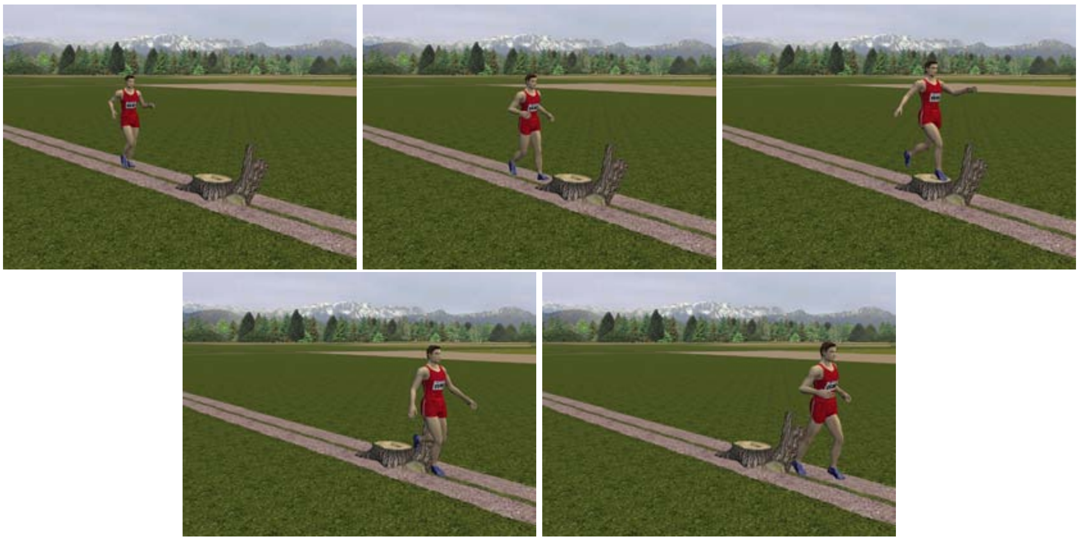


Figure 2.5: Sequences of a jump performed in the VHD++ platform.

Part V

Final Considerations

Chapter 1

Conclusion

In this chapter, we conclude first by summarizing the objectives and the results, and secondly by describing our contributions. Thirdly, we give outlook on further research directions and possible extensions.

1.1 Summary

The aim of this thesis was to propose an on-line locomotion engine for virtual humans so as to make them move and jump in an environment. In addition, the engine controls the clearing of possible dynamic obstacles in this environment. The directives guiding the engine conception were:

- to extend existing motion generation methods by developing a single animation system capable not only of generating motions, but also first of improving the quality results and secondly of adapting the motion to dynamic environments;
- to propose simple and efficiency methods at each level of our animation system in order to preserve real-time performances, motion realism and genericity;
- to give useful tools to animators or simulation processes for the animation generation.

Our approach was based on motion capture data composed of locomotion cycles and jumping sequences performed by various subjects. The representation of such motions has been simplified using a statistical method to facilitate the elaboration of a parametric model. This model has been structured into hierarchical levels in order to allow an intuitive motion parameterization: speed, type of locomotion and personification. Moreover, a normalization stage in time and space was introduced in order to adapt the produced animation to any character size. We also demonstrated the genericity of the motion model by applying it to cyclic motions (walking and running) as well as non-cyclic ones (jumping).

The guarantee of basic constraint preservation during the real-time motion synthesis has been ensured by firstly designing a new detection method for foot-floor interactions. Secondly, those constraints have been smoothly enforced using a novel mechanism.

The problem of assembling locomotion with jump actions was also dealt with. We introduced a method based on real observations of people performing jumps, which ensures a coherent on-the-fly synchronization between a required jump and the current locomotion parameters.

Finally, we have proposed a motion planning technique which reacts to obstacles in a dynamic environment, by automatically adopting a new trajectory. According to the parameter configuration, our method adapts the locomotion in order to jump over or to avoid the obstacle.

We have implemented those techniques in a single animation engine, capable of animating characters in an on-line manner. Two applications using this engine have been designed to alleviate the animator's work. In the first, animations can be generated by means of scripts describing motion parameters over time. The second is composed of a user interface to control the character animation in real-time.

1.2 Contributions

1.2.1 Motion Parameterization

Motion parameterization in Computer Animation is a wide topic still under exploration but already providing interesting results. However, the existing locomotion synthesis approaches lack in intuitive and/or accurate motion parameterization. In addition, they actually compute the final parameterized posture frame by frame, at each time-step.

We propose a new motion synthesis approach by structuring a motion capture database into hierarchical levels. At each level, a statistical method (PCA) is applied to the captured motions which are then grouped by a given high-level parameter characterizing them. In this way, we contributed to generate accurate motions at each level by an interpolation between motions having similar attributes. This scheme not only allows the production of new motions inside the convex hull determined by the original data, but also beyond it (extrapolation). To compute an entire posture set at once in a very efficient way, we proposed to apply PCA in order for the algorithm to consider the statistical variation between the complete motions instead of their individual frames. With similar performances to existing frame by frame methods, we proposed to compute the entire set of postures at once from a motion unit, like a locomotion cycle for example.

In the literature, motion models do not consider character size. All motions from the database therefore have to come from the same character. If characters of different sizes have to be animated, it is necessary to adjust the final motion with a retargeting method. Our model, however, is generic. It takes into account the character size and considers it as a high-level parameter. The motion is directly generated according to the proportions of the animated character. Besides this data normalization, our model is also generic in the sense that a large variety of motions can be parameterized, including cyclic locomotion (walk, run) and non-cyclic sequences (jump).

The results of this contribution have been published in several papers for international conferences and journal [Boulic et al., 2003; Glardon et al., 2004a,b; Urtasun et al., 2004].

1.2.2 Constraint Detection and Enforcement

A special constraint in character animation, namely the foot-floor interaction, is important to consider as its preservation increases the realism of generated motions. Thanks to the ability of our motion model to compute an entire motion unit and therefore to obtain postures in the future, we bring two complementary contributions in this context of foot constraints.

The first contribution concerns the detection of such a constraint in an on-line manner. Thanks to the available motion unit, we extended an off-line technique for its application to on-line motion synthesis. In order to replace the manual fine-tuning of control parameters required by off-line approaches, our technique automatically adapts those parameters with respect to properties of the current generated motion.

The second contribution consists in developing a constraint enforcement method which not only maintains a foot fixed during its constraint but also allows to re-position this foot at ground level, at a given location. Using the motion unit in order to anticipate the constrained foot position, the method provides a smooth enforcement before and after the constraint.

The combination of these two approaches has shown an improvement in the quality of final motions. This solution, by using future information of the motion, allows the modification of straight line locomotion so as to follow a curved path.

The results of this contribution is in review process for a journal submission [Glardon et al., 2006a].

1.2.3 Coherent Motion Transition

A number of approaches have been explored to generate transitions between different types of motions. Some aim at finding transition time automatically from a big motion database, in an off-line manner, while others attempt to synchronize two different actions in an on-line manner. However, the actions are always pre-defined and can not be parameterized during the animation run-time.

We contributed to improve the transition generation (i.e. blending) by proposing a technique which takes into account the different dynamic properties of the motions to be blended. Our technique is able to perform on-line transitions from a current locomotion to any jump type (and inversely) by adapting the locomotion so as to be compatible to the requested jump. This compatibility criterion consists in a functional model based on real motion observations.

In addition, the transition time and duration are automatically determined on-the-fly through our constraint detection method. Hence, our contribution greatly expands the animator freedom. In fact, from any locomotion characteristics and any parameterized jumps selected by an animator, our method automatically generates a coherent transition.

The results of this contribution have been published in [Glardon et al., 2005].

1.2.4 Dynamic Obstacle Handling

Previous motion planning techniques focus on solving collision-free paths in complex environments. They are interactive at the best case, but do not generate motions. Additionally, they are never adapted for dynamic environments where the obstacle configuration is unpredictable. While other approaches tend to be on-line, they can still not handle dynamic obstacles, and produce rough animation results.

We contributed to change this situation by presenting a method which mirrors, as closely as possible, the behavior of a human confronted with an obstacle to jump over. In fact, the humans react as soon as they see an obstacle, and they do not need to know the position of all obstacles in the environment. Hence, our method constructs the appropriate trajectory as soon as an obstacle is created in front of an animated character. The technique we propose computes the adequate speed profile to adopt along the path towards the obstacle. This profile ensures first that the take-off foot is placed as close as possible to the obstacle, and secondly, that the final run-up speed corresponds to the required jump. The obstacle can also be got round when it is too big to be jumped over.

Thanks to this contribution, it is feasible to animate on-line autonomous agents which are reactive to a dynamic environment. Together with the previously presented contributions for character animation, it is an appropriate means to precisely clear obstacles. More generally, our method is able to generate a continuous motion which corresponds to given initial and final conditions set on the locomotion speed and foot positions.

The results of this contribution have been accepted for publication ([Glardon et al., 2006b]).

1.2.5 Integration and Applications

The final contribution of this thesis consists in the integration of all the animation techniques we have proposed. The main difficulty was to preserve the real-time performance, while reaching a compromise between motion quality and on-line generation. Thanks to the appropriate tradeoff between efficiency and precision proposed in each of the previous contributions, we have implemented a complete animation engine fully controlled by high-level parameters and reactive to dynamic obstacles.

To test our final engine, a 3D application was developed. The animation of a character is possible through a user interface. At any time, the animator can generate an obstacle with random dimensions and place it in front of the character at a random distance. The engine reacts automatically by modifying the current locomotion parameters in order to clear the obstacle. A succession of randomly created obstacles provides a test bed, and allows for the observation and evaluation of the important aspects of our animation model: smooth high-level parameterization; constraint enforcement; take-off foot position; transition generation.

Finally, we contributed to the creation of a motion database composed of more than 500 minutes of animation. Besides its usefulness to the research community for motion analysis and synthesis, this raw material is valuable. On one hand, motion capture systems are expensive and the post-processing work to clean the final animations is prohibitive. On the other hand, only a few research labs or commercial companies open their motion databases to the public.

1.3 Perspectives

1.3.1 Result Improvements

The animation engine resulting from this thesis provides with an intuitive and continuous control of the generated motions, allied to the possibility of interaction with the environment of the animated character. However, some aspects of our method could be improved.

First, we emphasized the need for an on-line animation engine. Along this thesis, we reached the compromise between motion quality and efficiency for each component of our engine. Nevertheless, the constraint detection may provide results which are not precise enough for short time constraints. As a consequence, the constraint enforcement becomes infeasible and artifacts appear in the final corrected motion. This is essentially due to the truncation of the start and end constraint times. To minimize this problem, a solution consists in increasing the sampling rate of the motion.

Second, the parameterization with respect to the captured subject, namely the personification parameter, is difficult to use in our system, especially for the locomotion. The main causes are the use of a treadmill for the capture and the lack of diversity in our database. Several approches may improve the influence of this personification. Clearly, the capture of exaggerated locomotions performed by an actor would be a good initiative. Another option would be to modify the final motion by wielding the motion amplitude or the offset between limbs and body.

To conclude, our motion modeling is based on statistical and approximation methods. Besides its advantages (efficiency, simplicity), this method, by definition, needs an important number of motions. Even if we have obtained acceptable results reducing the number of examples, it degrades the guarantee of quality and accuracy of the results. The combination of our motion modeling with recent approaches, which interpret PCA as a particular Gaussian process, can be a method to restrict the number of necessary input data while keeping good quality results.

1.3.2 Concrete Applications

The development of our animation engine opens wide application perspectives. One of the most popular concerns the game and movie industry. In these domains, the current applied techniques combine motion capture with manual edition by keyframing. Despite its significant cost in time and money, such approach is still used as it provides the best quality and control for animations. The technology transfer from research to industry requires the organization of new work methods and new training for designers and animators. However, this transfer is becoming more and more effective, especially for animations with low level of details, like background character motions.

Simulation of human behaviors is another interesting application for our engine. For town planners, it is important to experiment their environment modifications by simulations giving rapid feedback, ideally on-line. In most cases, they need a large amount of people walking and running in the urban traffic, and jumping over small obstacles such as sidewalks, puddles of water or roadworks. Our method is very appropriate for parameterized animation of crowds thanks to the computation of a motion unit at once, and not frame by frame. In fact, when the character's motion parameters are not modified during the animation, no extra computation is needed. For crowd simulations, this approach allows for the reach of a compromise somewhere between continuous parameter variations of few characters, and fixed, keyframe based, animations of thousands of characters. The principle consists in updating, at each time step of the simulation, the motion parameters for only few characters.

Our motion modeling is also applicable for medical applications, especially in orthopedic domains. After undergoing a surgery, people have to follow rehabilitation exercises. These can be realized with special motorized prostheses which help the patient to walk, for example. Our motion modeling could control the motion of those prostheses. Another important aspect in rehabilitation is the motion comparison between an injured and a healthy person. By overlapping motions generated by our engine with the patient's ones, the physician has a technical and pedagogical tool to explain the correction the patient should bring.

1.3.3 Future Extensions

In the future, our animation engine can be extended particularly to provide more flexibility and autonomy.

In this thesis, we dealt with jumps of variable lengths. In reality, however, people perform a wider variety of jumps according to the environment they are confronted to. We can imagine jumps of variable heights, jumps without run-up or jumps with a level change between the take-off and landing positions. Modeling of such motions is possible using our methodology. For example, in case of jumps without run-up, our motion planning method is capable of determining the speed variation so as to stop the character exactly in front of the obstacle.

Another interesting extension is the combination of a perception system with our animation engine. This system could simulate the human perception to detect obstacles in the environment. After this detection, the obstacle is analyzed in order to determine the way the character will react. For example, if this obstacle can be jumped over, the jump type has to be chosen as well as its parameters.

Recently, the evaluation and validation of synthesized motions have emerged, and this research topic is in vogue. The underlying idea is to understand the human mechanism which determines whether a motion is realistic or not. At this time, specific type of motions can be evaluated by physically based validation. We can imagine a method which evaluates a wide spectrum of motions as well as transition between them. Such methods could be applied to our engine in order to validate and/or improve our approach to generate animations.

Appendix A

List of Notations

Throughout this document, scalar are denoted by small letters such as s, vectors by small boldface letters such as $\mathbf{v}$. Matrices are denoted by capital boldface letters such as $\mathbf{M}$.

Motions

Description	Notation
Motion unit	$\boldsymbol{\theta}$
Number of frames in a motion unit	N_{frame}
Continuous motion function (over time t)	$\boldsymbol{M}(t)$
i-th frame of a motion	$\mathcal{F}_i$
Motion phase	φ
Position of root joint in motion unit, at i-th frame	$\widehat{\mathbf{p}}_{ri}$
Orientation of root joint in moiton unit, at i-th frame	$\widehat{\mathbf{q}}_{ri}$
Position of root joint, at i-th frame	$\mathbf{p}_{ri}$
Orientation of root joint	$\mathbf{q}_{ri}$
Orientation of the j-th joint, at i-th framet	$\mathbf{q}_{ji}$

Motion Parameters

Description	**Notation**
Locomotion parameter vector	$\mathbf{\Psi} = (s, w_{loco}, \mathbf{w}_{subj})$
Speed	s
Locomotion weight	w_{loco}
Personification vector	$\mathbf{w}_{subj}$
Personification i-th element	$w_{subj,i}$
Angular speed	ω
Jump Length	l

Motion Capture Sequences

Description	**Notation**
Number of database subject	N_{subj}
Number of walking sequences	N_{seq}^{walk}
Number of running sequences	N_{seq}^{run}
Number of sequences of the k-th subject	N_{seq}^{k}
Number of walking sequences of the k-th subject	$N_{seq}^{walk,k}$
Number of running sequences of the k-th subject	$N_{seq}^{run,k}$
Minimum captured speed for walking	s_w
Minimum captured speed for running	s_r

PCA Spaces

Simple PCA Walking	**Components**	**Notation**
PCA (for the k-th subject)	Matrix of Principal Components	$\mathbf{E}^k$
	Vector of the j-th Principal Component	$\mathbf{e}_j^k$
	Coefficient Vector	$\boldsymbol{\alpha}^k$

Hierarchical PCA space Locomotion	**Components**	**Notation**
Main PCA	Matrix of Principal Components	$\mathbf{E}$
	Vector of the j-th Principal Component	$\mathbf{e}_j$
	Coefficient Vector	$\boldsymbol{\alpha}$
Sub-PCA level 1 (for the k-th subject)	Matrix of Principal Components	$\mathbf{F}^k$
	Vector of the j-th Principal Component	$\mathbf{f}_j^k$
	Coefficient Vector	$\boldsymbol{\beta}^k$
Sub-PCA level 2 (walk, for the k-th subject)	Matrix of Principal Components	$\mathbf{G}^{walk,k}$
	Vector of the j-th Principal Component	$\mathbf{g}_j^{walk,k}$
	Coefficient Vector	$\boldsymbol{\gamma}^{walk,k}$
	Function returning $\boldsymbol{\gamma}^{walk,k}$ given a speed s	$\boldsymbol{A}^{walk,k}(s)$
Sub-PCA level 2 (run, for the k-th subject)	Matrix of Principal Components	$\mathbf{G}^{run,k}$
	Vector of the j-th Principal Component	$\mathbf{g}_j^{run,k}$
	Coefficient Vector	$\boldsymbol{\gamma}^{run,k}$
	Function returning $\boldsymbol{\gamma}^{run,k}$ given a speed s	$\boldsymbol{A}^{run,k}(s)$

Hierarchical PCA space Walking Jump	**Components**	**Notation**
Main PCA	Matrix of Principal Components	$\mathbf{E}$
	Vector of the j-th Principal Component	$\mathbf{e}_j$
	Coefficient Vector	$\boldsymbol{\alpha}$
Sub-PCA level 1 (for the k-th subject)	Matrix of Principal Components	$\mathbf{F}^k$
	Vector of the j-th Principal Component	$\mathbf{f}_j^k$
	Coefficient Vector	$\boldsymbol{\beta}^k$
	Function returning $\boldsymbol{\beta}^k$ given a jump length l	$\boldsymbol{A}^{jmpW,k}(l)$

Hierarchical PCA space Running Jump	**Components**	**Notation**
Main PCA	Matrix of Principal Components	$\mathbf{E}$
	Vector of the j-th Principal Component	$\mathbf{e}_j$
	Coefficient Vector	$\boldsymbol{\alpha}$
Sub-PCA level 1 (for the k-th subject)	Matrix of Principal Components	$\mathbf{F}^k$
	Vector of the j-th Principal Component	$\mathbf{f}_j^k$
	Coefficient Vector	$\boldsymbol{\beta}^k$
	Function returning $\boldsymbol{\beta}^k$ given a jump length l	$\boldsymbol{A}^{jmpR,k}(l)$

Frequency Functions

Description	**Notation**
Motion frequency function	F
Locomotion frequency function	F_{loco}
Locomotion frequency function of the k-th subject	F_{loco}^{k}
Locomotion (walk) frequency function of the k-th subject	$F_{loco}^{walk,k}$
Locomotion (run) frequency function of the k-th subject	$F_{loco}^{run,k}$
Jump frequency function	F_{jump}
Jump frequency function of the k-th subject	F_{jump}^{k}
Jump (walk) frequency function of the k-th subject	$F_{jump}^{walk,k}$
Jump (run) frequency function of the k-th subject	$F_{jump}^{run,k}$

Appendix B

Background Mathematics

In this appendix, we present the mathematical background used in this thesis. The first section introduces basic statistical definitions. The second presents two matrix decomposition used in linear algebra. In the last section, the Principal Component Analysis (PCA) method is described.

B.1 Basic Statistics

This section contains the definitions and rules for the mean, variance, standard deviation, covariance and correlation for a given sample dataset $\mathbf{X} = (x_1, x_2, \ldots, x_n)$ extracted from a population. Note that the definitions have been stemmed from [Bronstein et al., 1995].

B.1.1 Mean

The arithmetic **mean**, or more specifically the population mean, μ_X of $\mathbf{X}$, is defined as follows:

$$\mu_X = \frac{1}{n}\sum_{i=1}^{n} x_i = E(\mathbf{X}) \tag{B.1}$$

The rules for the mean are:

$$E(a\mathbf{X} + b) = aE(\mathbf{X}) + b \tag{B.2}$$

$$E(\mathbf{X} + \mathbf{Y}) = E(\mathbf{X}) + E(\mathbf{Y}) \tag{B.3}$$

B.1.2 Variance

The **variance** σ_X^2 of $\mathbf{X}$ is defined as follows:

$$\begin{aligned}\sigma_X^2 &= \frac{1}{n}\sum_{i=1}^{n} x_i - \mu_X = E\left(x_i - E(\mathbf{X})^2\right)\\ &= E\left(x_i^2 - 2x_i E(\mathbf{X}) + E(\mathbf{X})^2\right) = E\left(\mathbf{X}^2\right) - E(\mathbf{X})^2\\ &= \mathit{VAR}(\mathbf{X}) \end{aligned} \tag{B.4}$$

The rules for the variance are:

$$\mathit{VAR}(a\mathbf{X}+b) = a^2\mathit{VAR}(\mathbf{X}) \tag{B.5}$$

$$\mathit{VAR}(\mathbf{X}+\mathbf{Y}) = \mathit{VAR}(\mathbf{X}) + 2\mathit{VAR}(\mathbf{X},\mathbf{Y}) + \mathit{VAR}(\mathbf{Y}) \tag{B.6}$$

where the rule B.6 is only true if $\mathbf{X}$ and $\mathbf{Y}$ are two independent variables. variables.

B.1.3 Standard Deviation

The **standard deviation** σ_X of $\mathbf{X}$ is defined as follows:

$$\sigma_X = \sqrt{\sigma_X^2} = \sqrt{\mathit{VAR}(\mathbf{X})} = \mathit{SD}(\mathbf{X}) \tag{B.7}$$

B.1.4 Covariance

The **covariance** σ_{XY} between $\mathbf{X}$ and $\mathbf{Y}$ is defined as follows:

$$\begin{aligned}\sigma_{XY} &= \sum_{i=1}^{n}\left(x_i - E(\mathbf{X})\right)\left(y_i - E(\mathbf{Y})\right) = E\left(\left(x_i - E(\mathbf{X})\right)\left(y_i - E(\mathbf{Y})\right)\right)\\ &= E(x_i y_i - x_i E(Y) - y_i E(X) + E(\mathbf{X})E(\mathbf{Y})) \qquad (B.8)\\ &= E(\mathbf{XY}) - E(\mathbf{X})E(\mathbf{Y}) = \mathit{COV}(\mathbf{X},\mathbf{Y}) \qquad (B.9)\end{aligned}$$

The covariance indicates the degree of association of two variables $\mathbf{X}$ and $\mathbf{Y}$. If σ_{XY} is positive, the relation between these two variables is increasing, and inversely when σ_{XY} is negative. If the measurement units of the two variables are different, the multiplication of those units returns the unit of the covariance. Therefore, the covariance is sensible to the unit changes.

B.1.5 Correlation

The **coefficient correlation** ρ_{XY} between $\mathbf{X}$ and $\mathbf{Y}$ is defined as follows:

$$\rho_{XY} = \frac{\mathit{COV}(\mathbf{X},\mathbf{Y})}{\sqrt{\mathit{VAR}(\mathbf{X})\,\mathit{VAR}(\mathbf{Y})}} = \mathit{CORR}(\mathbf{X},\mathbf{Y}) \tag{B.10}$$

The correlation, defined between -1 and 1, is the normalized covariance and does not have unit. The correlation, in contrast to the covariance, does not vary when the measurement units of the data are changing.

B.2 Matrix Algebra

This section focuses on a brief overview of the eigenvalue decomposition and the singular value decomposition to introduce the concept of PCA. Note that the definitions have been stemmed from three books [Nipp and Stoffer, 1992; Press et al., 1992; Jolliffe, 1986].

B.2.1 Eigenvalue Decomposition

Let $\mathbf{A}$ be a quadratic matrix of type $(n \times n)$. The classical mathematical eigenvalue problem is defined as the solution of the following equation:

$$\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i \qquad i = 1, 2, \ldots, n \tag{B.11}$$

where $\mathbf{v}_i$ are the eigenvectors and λ_i the corresponding eigenvalues of $\mathbf{A}$. If the matrix $\mathbf{A}$ is symmetric $(\mathbf{A} = \mathbf{A}^T)$, the following properties can be derived:

1. $\mathbf{A}$ has exactly n eigenvalues.
2. The eigenvectors build an orthogonal basis.
3. $\mathbf{A}$ can be decomposed into a diagonal matrix $\mathbf{D}$ whose elements contain the eigenvalues of $\mathbf{A}$, and an orthogonal matrix $\mathbf{U}(i.e.\mathbf{U}^T\mathbf{U} = \mathbf{I})$ whose rows contain the corresponding unit eigenvectors. The decomposition is described as:

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{U}^T = \mathbf{U}\mathbf{D}\mathbf{U}^{-1} \tag{B.12}$$

B.2.2 Singular Value Decomposition

Singular Value Decomposition (SVD) methods are based on the following theorem of linear algebra:

Theorem 1 *Any $(m \times n)$ matrix $\mathbf{A}$ whose number of rows m is greater than or equal to its number of columns n can be written as the product of an $m \times n$ column-orthogonal matrix $\mathbf{V}$, an $n \times n$ diagonal matrix $\mathbf{\Sigma}$ with positive or zero elements (the singular values), and the transpose of an $n \times n$ orthogonal matrix $\mathbf{U}$.*

The singular values d_i hold in $\mathbf{\Sigma}$ are defined as follows:

$$d_i = \sqrt{\lambda_i} \qquad i = 1, 2, \ldots, n \tag{B.13}$$

where the λ_i are the eigenvalues of the matrix $\mathbf{A}^T\mathbf{A}$. Their corresponding eigenvector $\mathbf{u}_i$ are called the right singular vectors of $\mathbf{A}$, and the corresponding eigenvector $\mathbf{v}_i$ of $\mathbf{A}\mathbf{A}^T$ the left singular vectors. Thereby the matrix $\mathbf{A}^T\mathbf{A}$ holds the same $\lambda_i(\neq 0)$ eigenvalues of $\mathbf{A}\mathbf{A}^T$:

$$\begin{aligned} \mathbf{A}^T\mathbf{A}\mathbf{u}_i &= \lambda_i\mathbf{u}_i \\ \mathbf{A}\mathbf{A}^T\mathbf{v}_i &= \lambda_i\mathbf{v}_i \end{aligned} \tag{B.14}$$

Furthermore, we can write:

$$\begin{aligned} \mathbf{A}\mathbf{u}_i &= d_i\mathbf{v}_i = \sqrt{\lambda_i}\mathbf{v}_i \\ \mathbf{A}^T\mathbf{v}_i &= d_i\mathbf{u}_i = \sqrt{\lambda_i}\mathbf{u}_i \end{aligned} \tag{B.15}$$

It is to observe that $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}\mathbf{A}^T$ are symmetric matrices and therefore have orthogonal eigenvectors matrix, $\mathbf{U}$ and $\mathbf{V}$. The SVD performed on the matrix $\mathbf{A}$ is defined as follows:

$$\mathbf{A} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T \tag{B.16}$$

B.2.3 PCA on Single Dataset

Suppose that $\mathbf{x}$ is a vector of p random variables, and that the variance of the p random variables and the structure of the covariance or correlations between the p variables are of interest. By applying a PCA on this vector, we have first to look for a linear function $\boldsymbol{\alpha}_1^T\mathbf{x} = z_1$ which has a maximum variance. Next we look for a linear function $\boldsymbol{\alpha}_2^T\mathbf{x} = z_2$, uncorrelated with $\boldsymbol{\alpha}_1^T\mathbf{x} = z_1$, which has a maximum variance, and so on. The $\boldsymbol{\alpha}_i^T\mathbf{x}$ correspond to the coefficient of the i-th principal component (PC). The following equation describes the computation of the first PC component z_1:

$$\boldsymbol{\alpha}_1^T\mathbf{x} = \begin{pmatrix} \alpha_{11} & \alpha_{12} & \dots & \alpha_{1p} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix} = \sum_{i=1}^{p} \alpha_{1i}x_i = z_1 \tag{B.17}$$

In order to find the value $\boldsymbol{\alpha_i}$, the computation of the variance of $\boldsymbol{\alpha}_i\mathbf{x}$ is described in Eq. B.18 by using rules of Eq.B.5. The maximization of the variance is performed using the technique of Lagrange multipliers (see [Jolliffe, 1986] for further details). As a result, the vector $\boldsymbol{\alpha}_i$ are the eigenvectors of the covariance matrix of $\mathbf{x}$, $\boldsymbol{\alpha}_i$ corresponding to the largest eigenvalue, and so on.

$$\begin{aligned} VAR\left(\boldsymbol{\alpha}_1\mathbf{x}\right) &= VAR\left(\sum_{i=1}^{p}\alpha_{1i}x_i\right) \\ &= \sum_{i=1}^{p}\alpha_{1i}^2\sigma_i^2 + 2\left(\sum_{i=1}^{p}\sum_{j=i+1}^{p}\alpha_{1i}\alpha_{1j}\sigma_{ij}\right) = \boldsymbol{\alpha}_\mathbf{1}^T COV\left(\mathbf{x}\right)\boldsymbol{\alpha}_\mathbf{1} \end{aligned} \tag{B.18}$$

B.2.4 PCA on Sample Dataset

Concerning the problem based on sample dataset, we replace the vector $\mathbf{x}$ by the matrix $\mathbf{A}$ containing in each row an observation $\mathbf{A}_i$. The number of column in $\mathbf{A}$ represents therefore the number of variables of the dataset (p for example), and the number of row corresponds to the number of samples (n for example). We look for the linear function $\mathbf{A}\boldsymbol{\alpha}_1 = \mathbf{z}_1$ which has a maximum variance. Comparing to the case of single dataset, the difference is that the coefficient of the PC is a vector containing the coefficient for every sample. Eq B.19 defines the coefficient vector $\mathbf{z}_1$ of the first PC for every sample.

$$\mathbf{A}\boldsymbol{\alpha}_1 = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{np} \end{pmatrix} \begin{pmatrix} \alpha_{11} \\ \alpha_{11} \\ \dots \\ \alpha_{1p} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{p} \alpha_{1i} a_{1i} \\ \sum_{i=1}^{p} \alpha_{1i} a_{2i} \\ \dots \\ \sum_{i=1}^{p} \alpha_{1i} a_{ni} \end{pmatrix} = \begin{pmatrix} z_{11} \\ z_{12} \\ \dots \\ z_{1n} \end{pmatrix} = \mathbf{z}_1 \tag{B.19}$$

In order to find the value α_{1i}, we define the variance σ_k^2 of a variable at the k-th column of the matrix $\mathbf{A}$ as follows:

$$\sigma_k^2 = \frac{1}{n-1} \sum_{i=1}^{n} (a_{ik} - \bar{a_k})^2 \tag{B.20}$$

where

$$\bar{a_k} = \frac{1}{n} \sum_{i=1}^{n} a_{ik} \qquad k = 1, 2, \dots, p \tag{B.21}$$

The maximization of those variances leads to a covariance matrix $\mathbf{C}$ whose eigenvectors return the α_i vectors. The covariance between the j-th variable and the k-th variable is defined as follows:

$$C_{ij} = \frac{1}{n-1} \sum_{i=1}^{n} (a_{ij} - \bar{a_j})(a_{ik} - \bar{a_k}) \tag{B.22}$$

Thus, the matrix $\mathbf{C}$ can be written as

$$\mathbf{C} = \frac{1}{n-1} \widetilde{\mathbf{A}}^T \widetilde{\mathbf{A}} \tag{B.23}$$

where $\widetilde{\mathbf{A}}$ is a $(n \times p)$ matrix with (i, j)-th element $(a_{ij} - \bar{a_j})$. This representation is very useful because the eigenvectors of $\mathbf{C}$ are the same as $\widetilde{\mathbf{A}}^T \widetilde{\mathbf{A}}$ and the eigenvalues differ from a factor of $\frac{1}{n-1}$.

To summarize, the eigenvectors $\mathbf{e}_i$ have to be computed from the covariance matrix $\mathbf{C}$, by the use of SVD methods for example. Hence, a matrix $\mathbf{E}$ is defined with its columns which contain the eigenvectors $\mathbf{e}_i$ classified in decreasing order. The general form of PCA applied to the matrix $\mathbf{A}$ can be written as:

$$\widetilde{\mathbf{A}}\mathbf{E} = \mathbf{Z} \tag{B.24}$$

In addition, as

$$\widetilde{\mathbf{A}} = \mathbf{A} - \bar{\mathbf{A}} \tag{B.25}$$

the matrix $\mathbf{A}$ can be defined as:

$$\widetilde{\mathbf{A}} = \bar{\mathbf{A}} + \mathbf{Z}\mathbf{E}^T \tag{B.26}$$

Appendix C

Fit Functions on PCA Spaces

C.1 Walking and Running Motion Units

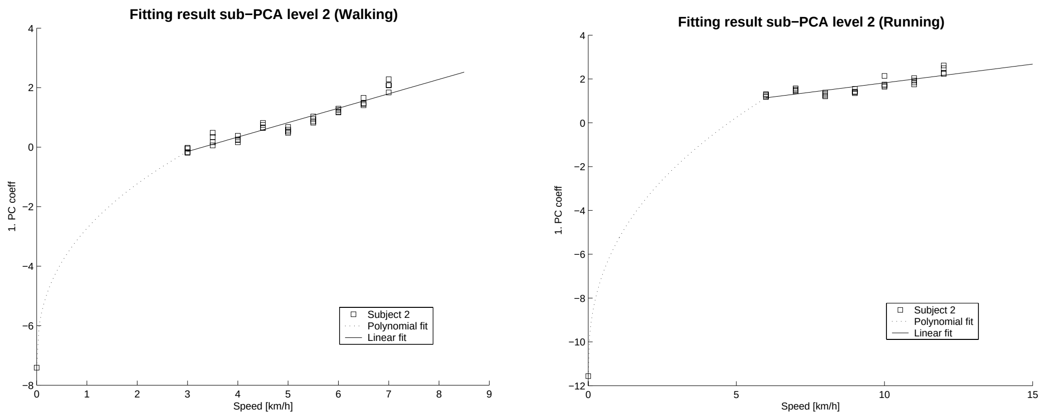


Figure C.1: Resulting fit functions for walking (left) and running (right) for the subject 2.

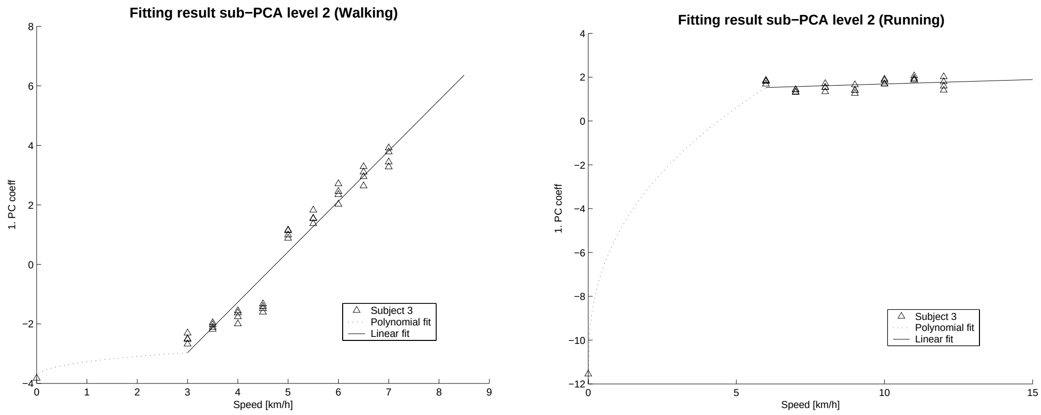


Figure C.2: Resulting fit functions for walking (left) and running (right) for the subject 3.

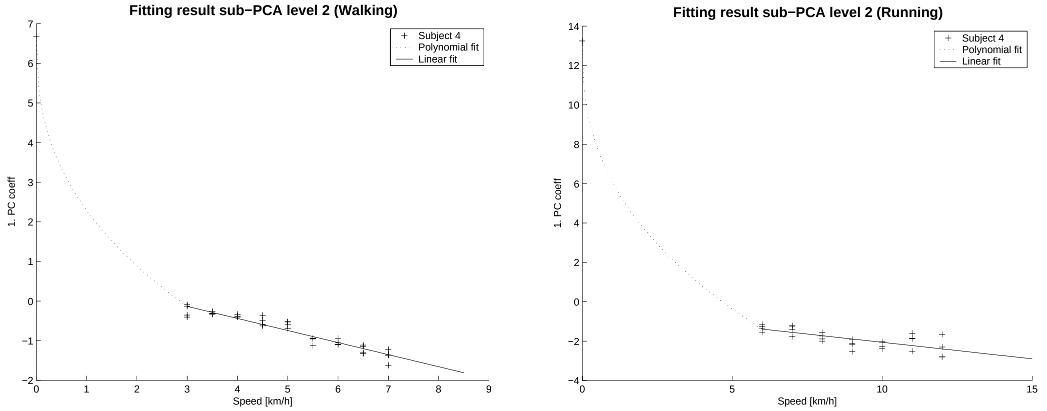


Figure C.3: Resulting fit functions for walking (left) and running (right) for the subject 4.

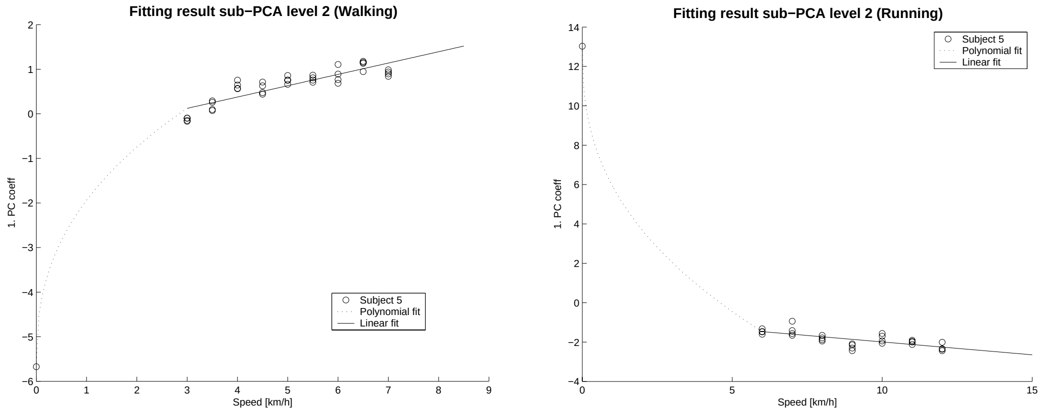


Figure C.4: Resulting fit functions for walking (left) and running (right) for the subject 5.

C.2 Walking and Running Jump Motion Units

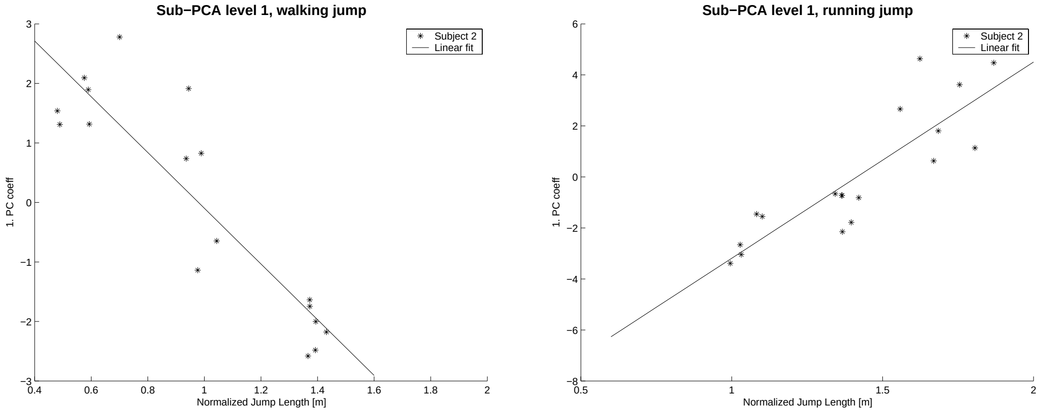


Figure C.5: Resulting fit function for walking (left) and running (right) jump for the subject 2.

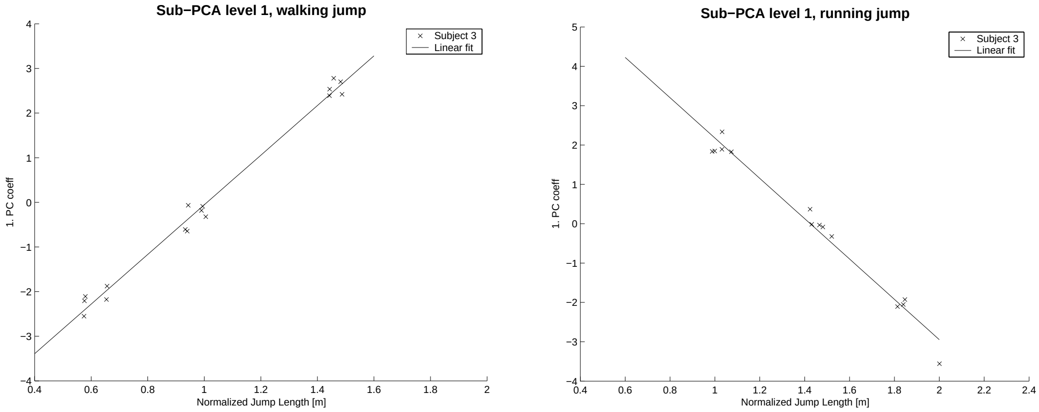


Figure C.6: Resulting fit function for walking (left) and running (right) jump for the subject 3.

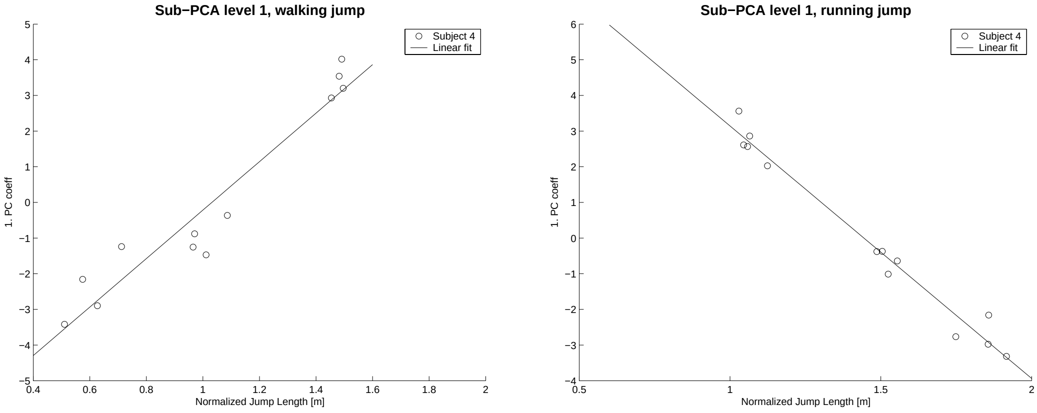


Figure C.7: Resulting fit function for walking (left) and running (right) for the subject 4.

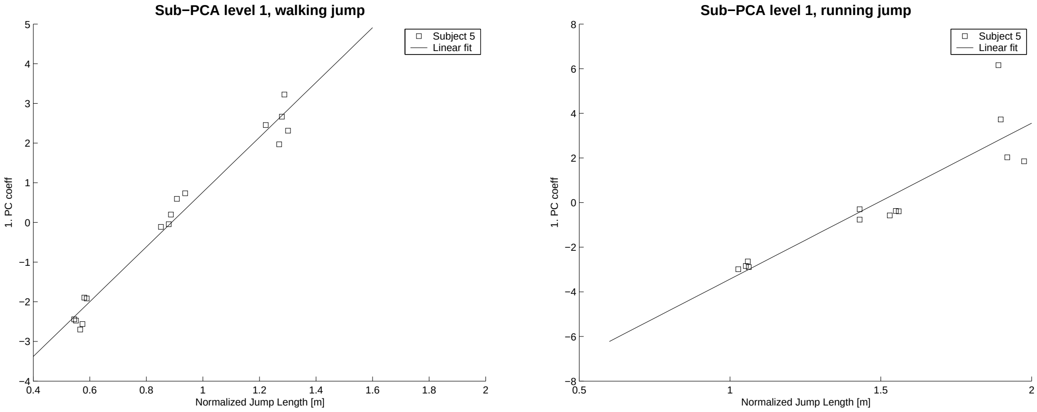


Figure C.8: Resulting fit function for walking (left) and running (right) for the subject 5.

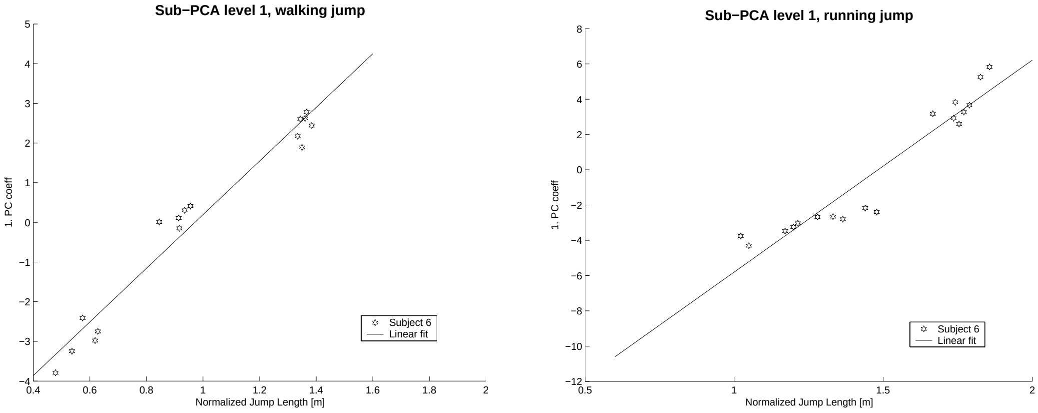


Figure C.9: Resulting fit function for walking (left) and running (right) for the subject 6.

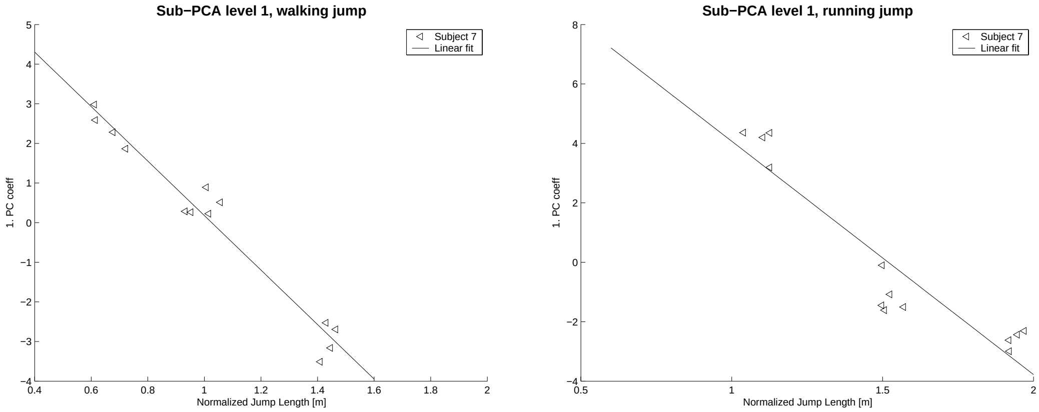


Figure C.10: Resulting fit function for walking (left) and running (right) for the subject 7.

Appendix D

Pecub’s Cartoons

The humoristic illustrations appearing in this thesis have been drawn by Pécub, a renowned cartoonist located in Aubonne, Switzerland. The illustrations used for the private defense and public presentation of this thesis are presented below. The author of this thesis holds the copyright of all cartoons, a present of his parents who are good friends of Pécub.

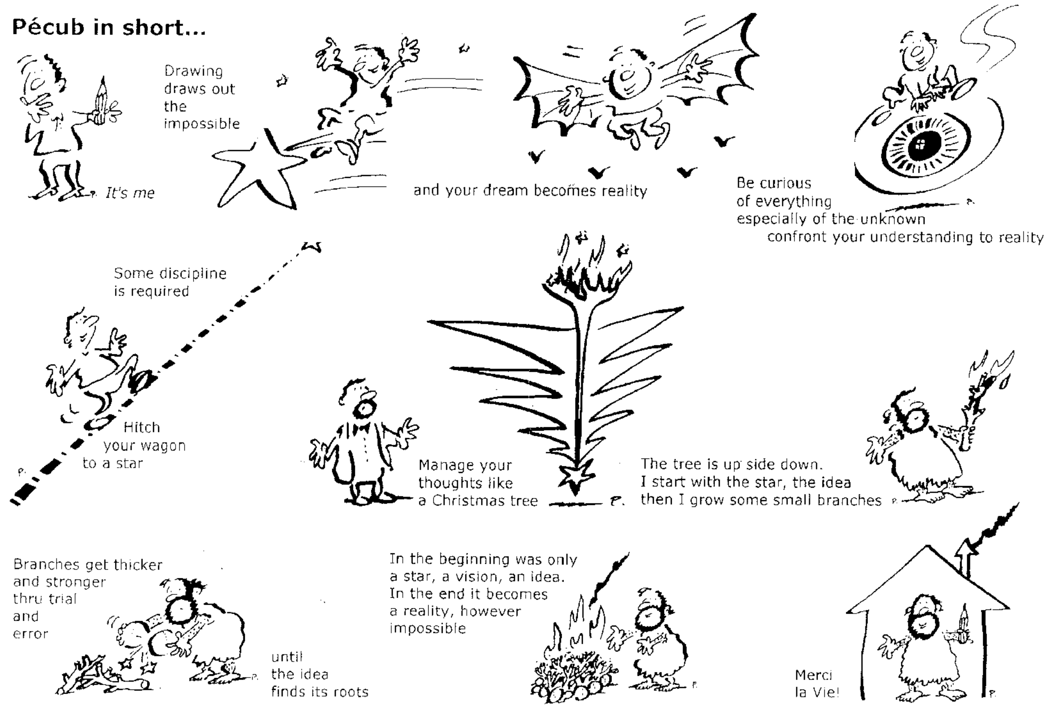

Pécub in short...
Drawing draws out the impossible
It's me
and your dream becomes reality
Be curious of everything especially of the unknown
confront your understanding to reality
Some discipline is required
Hitch your wagon to a star
Manage your thoughts like a Christmas tree
The tree is up side down.
I start with the star, the idea
then I grow some small branches
Branches get thicker and stronger thru trial and error
until the idea finds its roots
In the beginning was only a star, a vision, an idea.
In the end it becomes a reality, however impossible
Merci la Vie!

Figure D.1: PCA performed on angular data instead of 3D marker positions (see Part II, Section 2.2).

Figure D.2: Time normalization to determine the duration of a generated motion (see Part III, Section 2.2).

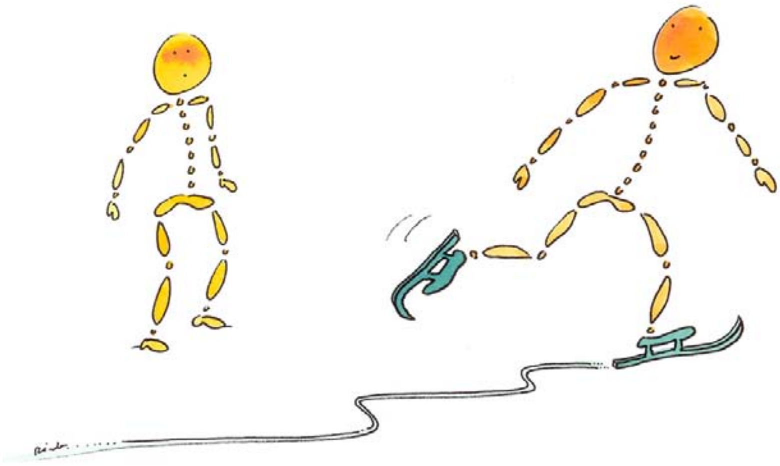


Figure D.3: Foot sliding effect which has to be corrected (see Part III, Chapter 2).

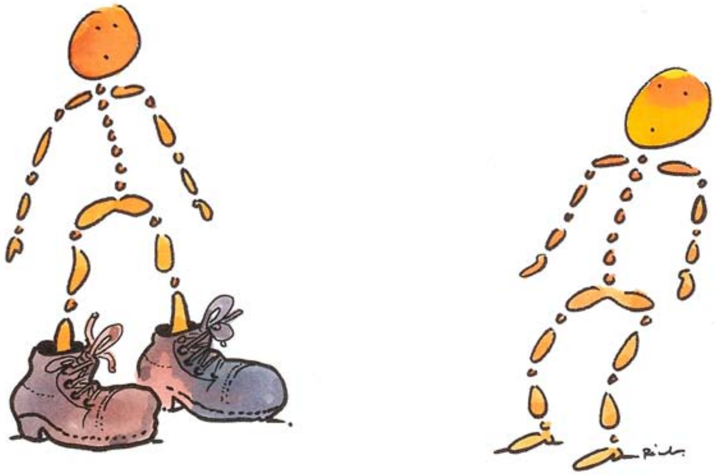


Figure D.4: Footplant detection method (see Part III, Section 2.3).

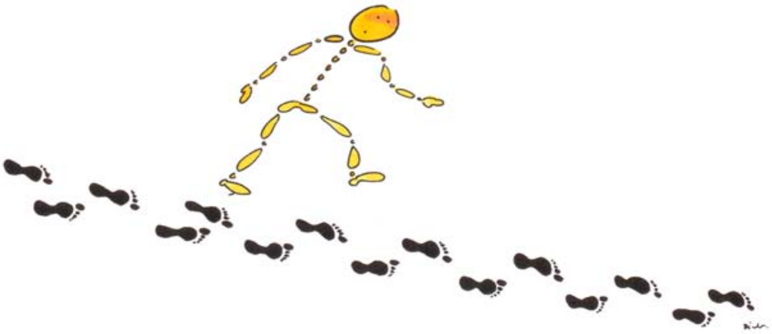


Figure D.5: Footprint position computation to place the take-off foot as close as possible to the obstacle (see Part IV, Section 1.3.2).

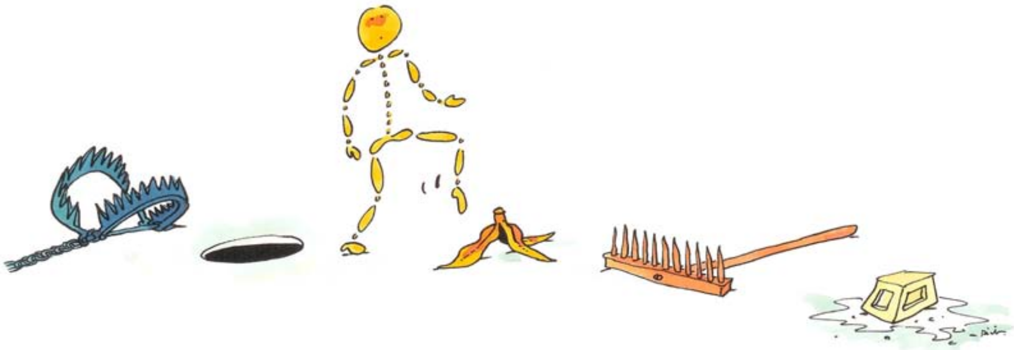


Figure D.6: Improvement of our method to dynamically handle various obstacle type (see Part IV, Section 1.6).

Figure D.7: Limitations of the methods presented in this thesis (see Part V, Section 1.3).

Figure D.8: Future perspectives with other activities which can be parameterized with our motion modeling (see Part V, Section 1.3).

Bibliography

3ds Max®. Autodesk. www.autodesk.com, 2005. 2.1.1

Y. Abe, C. Liu, and Z. Popović. Momentum-based parameterization of dynamic character motion. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004. 2.5

A. Ahmed, F. Mokhtarian, and A. Hilton. Cyclification of human motion for animation synthesis. In *Proceedings of Eurographics (Short Paper)*, sept 2003. 1.1

M. Alexa. Linear combination of transformations. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 380–387, 2002. 2.3.2, 1.2

M. Alexa and W. Müller. Representing animations by principal components. In *Proceedings of Eurographics*, pages 411–426, 2000. 2.3.2, 1.2

B. Allen, B. Curless, and Z. Popović. Articulated body deformation from range scan data. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2002. 2.3.1

K. Amaya, A. Bruderlin, and T. Calvert. Emotion from motion. In *Graphics Interface '96*, pages 222–229, 1996. 2.1.3

O. Arikan and D. Forsyth. Interactive motion generation from examples. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2002. 2.5, 2.5, 3.1, 3.4

O. Arikan, D. Forsyth, and J. O'Brien. Motion synthesis from annotations. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 402–408, 2003. 2.3.2, 2.18, 2.5

G. Ashraf and K. Wong. Generating consistent motion transition via decoupled framespace interpolation. In *Proceedings of Eurographics*, 2000. 2.5

N. Badler, R. Bindiganavale, J. Granieri, S. Wei, and X. Zhao. Posture Interpolation with Collision Avoidance. In *Proceedings of Computer Animation*, 1996. 2.6

P. Baerlocher and R. Boulic. An inverse kinematic architecture enforcing an arbitrary number of strict priority levels. *The Visual Computer*, 20(6):402–417, 2004. 2.4.2, 2.4

A. Berthoz. *The Brain's Sense of Movement*. Havard University Press, 2000. 2.1

R. Bindiganavale and N. Badler. Motion abstraction and mapping with spatial constraints. *Lecture Notes in Computer Science*, 1537:70–83, 1998. 2.4.1, 2.1

V. Blanz and T. Vetter. A morphable model for the synthesis of 3d faces. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 187–194, 1999. 2.3.2

V. Blanz, C. Basso, T. Poggio, and T. Vetter. Reanimating faces in images and video. In *Proceedings of Eurographics*, 2003. 2.3.2

V. Blanz, K. Scherbaum, T. Vetter, and H.-P. Seidel. Exchanging faces in images. In *Proceedings of Eurographics*, 2004. 2.3.2

R. Boulic. Proactive Steering Toward Oriented Targets. In *Proceedings of Eurographics, short presentation*, 2005. 1.3.1

R. Boulic, D. Thalmann, and N. Magnenat-Thalmann. A global human walking model with real time kinematic personification. *Visual Computer*, 6(6):344–358, 1990. 2.1.2, 1.3

R. Boulic, P. Becheiraz, L. Emering, and D. Thalmann. Integration of motion control techniques for virtual human and avatar real-time animation. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 111–118, 1997. 2.5

R. Boulic, P. Glardon, and D. Thalmann. From measurements to model: the walk engine. In *Proceedings of Optical 3-D Measurement Techniques VI*, pages 167–174, sept 2003. 1.2.1

R. Boulic, B. Ulciny, and D. Thalmann. Versatile walk engine. *Journal Of Game Development*, 1(1):29–50, 2004. 2.1.2, 1.3

M. Brand and A. Hertzmann. Style machines. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 183–192, 2000. 2.3.2, 2.12

C. Bregler, L. Loeb, E. Chuang, and H. Deshpande. Turning to the masters: Motion capturing cartoons. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 399–407, 2002. 2.3.2, 2.13

O. Brock and O. Khatib. Elastic Strips: A Framework for Motion Generation in Human Environments. *International Journal of Robotic Research*, 21(12):1031–1052, 2002. 2.6

I. Bronstein, K. Semendjajew, G. Musiol, and H. Mühlig. *Taschenbuch der Mathematik*. Harri Deutsch, 1995. B.1

A. Bruderlin and T. Calvert. Goal-directed, dynamic animation of human walking. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 233–242, 1989. 2.1.2

A. Bruderlin and T. Calvert. Interactive animation of personalized human locomotion. In *Proceedings of Graphics Interface*, pages 17–23, 1993. 2.1.2

A. Bruderlin and T. Calvert. Knowledge-driven, interactive animation of human running. In *Graphics Interface ’96*, pages 213–221, May 1996. 2.1.2, 1.3.1, 1.3.1

A. Bruderlin and L. Williams. Motion signal processing. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 97–104, aug 1995. 2.1.3, 2.5, 2.3.1, 2.5, 1.3, 2.4.1

P. Burt and E. Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, 1983. 2.1.3

S. Buss and J. Fillmore. Spherical averages and applications to spherical splines and interpolation. *ACM Transactions on Graphics*, 20(2):95–126, 2001. 1.5.1

M. Butz, O. Sigaud, and P. Gerard. Anticipatory Behavior in Adaptive Learning Systems. *Springer Verlag*, 2003. 2.4.3

Carrara®. Eovia corporation. www.eovia.com, 2005. 2.1.1, 2.1

J. Chai and J. Hodgins. Performance animation from low-dimensional control signals. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2005. 2.1.3

S. Chenney and D. Forsyth. Sampling plausible solutions to multi-body constraint problems. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 219–228, 2000. 2.3.2

J. Chestnutt, M. Lau, G. Cheung, J. Kuffner, J. Hodgins, and T. Kanade. Footstep Planning for the Honda ASIMO Humanoid. In *Proceedings of IEEE International Conference on Robotics and Automation*, apr 2005. 2.6

K. Choi and H. Ko. Online Motion Retargetting. *Journal of Vizualisation and Computer Animation*, 11:223–235, 2000. 2.4.2

M. Choi, J. Lee, and S. Shin. Planning Biped Locomotion using Motion Capture Data and Probabilistic Roadmaps. *ACM Transactions on Graphics*, 2003. 2.20, 2.6, 1.1, 1.3.2

S. Chung and J. Hahn. Animation of Human Walking in Virtual Environments. In *Proceedings of Computer Animation*, 1999. 2.1.2, 2.2

M. Cohen. Interactive spacetime constrol for animation. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 293–302, 1992. 2.1.3

T. Conde and D. Thalmann. An artificial life environment for autonomous virtual agents with multi-sensorial and multi-perceptives features. *Compute Animation and Virtual World*, 15: 311–318, 2004. 2.4.3

J. Dingwell, J. Cusumano, P. Cavanagh, and D. Sternad. Local dynamic stability versus kinematic variability of continuous overground and treadmill walking. *Journal of Biomechanical Engineering*, 123(1):27–32, feb 2001. 1.1

A. Egges, T. Molet, and N. Magnenat-Thalmann. Personalised real-time idle motion synthesis. In *Proceedings of Pacific Graphics*, 2004. 2.3.2

P. Faloutsos, M. van de Panne, and D. Terzopoulos. Composable controllers for physics-based character animation. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 251–260, 2001a. 2.1.2

P. Faloutsos, M. van de Panne, and D. Terzopoulos. The virtual stuntman: dynamic characters with a repertoire of autonomous motor skills. *Computer & Graphics*, 25(6):933–953, 2001b. 2.1.2, 2.3

P. Faloutsos, M. van de Panne, and D. Terzopoulos. Autonomous reactive control for simulated humanoids. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2003. 2.1.2

A. Fang and N. Pollard. Efficient synthesis of physically valid human motion. *ACM Transactions on Graphics*, 22(3):417–426, 2003. 2.1.3

K. Forbes and E. Fiume. An efficient search algorithm for motion data using weighted pca. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, aug 2005. 2.3.1

M. Girard. Interactive design of 3-D computer-animated legged animal motion. In *Proceedings of ACM Symposium on Interactive 3D Graphics*, pages 131–150, 1987. 2.1.2

M. Girard and A. Maciejewski. Computational Modeling for the Computer Animation of Legged Figures. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 263–270, jul 1985. 2.1.2, 2.4.2

P. Glardon, R. Boulic, and D. Thalmann. PCA-based walking engine using motion capture data. In *Proceedings of Computer Graphics International*, pages 292–298, 2004a. 1.2.1

P. Glardon, R. Boulic, and D. Thalmann. A coherent locomotion engine extrapolating beyond experimental data. In *Proceedings of Computer Animation and Social Agent*, pages 73–83, 2004b. 1.2.1

P. Glardon, R. Boulic, and D. Thalmann. On-line adapted transition between locomotion and jump. In *Proceedings of Computer Graphics International*, pages 44–49, 2005. 1.2.3

P. Glardon, R. Boulic, and D. Thalmann. Robust on-line adaptive footplant detection and enforcement for locomotion. *To appear in The Visual Computer*, 2006a. 1.2.2

P. Glardon, R. Boulic, and D. Thalmann. Dynamic obstacle clearing for real-time character animation. *To appear in The Visual Computer (march 2006)*, 2006b. 1.2.4

M. Gleicher. Comparing constraint-based motion editing methods. *Graphical Models*, 63 (2):107–134, mar 2001a. 2.1.3

M. Gleicher. Motion editing with spacetime constraints. In *Proceedings of ACM Symposium on Interactive 3D Graphics*, pages 139–148, apr 1997. 2.1.3

M. Gleicher. Retargetting motion to new characters. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 33–42, aug 1998. 2.6, 2.1.3

M. Gleicher. Motion path editing. In *Symposium on Interactive 3D Graphics*, pages 195–202, 2001b. 2.1.3

M. Gleicher, H. Shin, L. Kovar, and A. Jepsen. Snap together motion. In *Proceedings of ACM Symposium on Interactive 3D Graphics*, pages 181–188, 2003. 2.5

J. Go, T. Vu, and J. Kuffner. Autonomous Behaviors for Interactive Vehicle Animations. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2004. 2.6, 1.1

A. Golam and K. Wong. Dynamic time warp based framespace interpolation for motion editing. In *Graphics Interface*, pages 45–52, May 2000. 2.3.1, 2.5

J. Gonzàlez, J. Varona, F. Roca, and J. Villanueva. A comparison framework for walking performances using aspaces. *ELCVIA*, 5(3):105–116, 2005. 2.3.2

F. Grassia. Practical parameterization of rotations using the exponential map. *The Journal of Graphics Tools*, 3(3):29–48, 1998. 2.3.2, 2.2, 2.2

K. Grochow, S. Martin, A. Hertzmann, and Z. Popović. Style-based inverse kinematics. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2004. 2.3.2, 2.4.2

S. Guo and J. Robergé. A high-level control mechanism for human locomotion based on parametric frame space interpolation. In *Proceedings of Eurographics Workshop on Computer Animation and Simulation*, pages 95–107, sept 1996. 2.3.1, 2.5, 1.3

Gypsy 4®. Meta motion. www.metamotion.com, 2005. 2.4

H-ANIM. Humanoid animation working group. www.hanim.org, 2005. 2.4.2, 2.2, 2.3

J. Hodgins and N. Pollard. Adapting simulated behaviors for new characters. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 153–162, 1997. 2.1.2

J. Hodgins, W. Wooten, D. Brogan, and J. O'Brien. Animating human athletics. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 71–78, 1995. 2.1.2

A. Hreljac and R. Marshall. Algorithms to determine event timing during normal walking using kinematic data. *Journal of Biomechanics*, 33(6):783–786, jun 2000. 2.4.1

D. Hsu, R. Kindel, J.C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving. *International Journal of Robotic Research*, 21(3):233–255, 2002. 1.1

E. Hsu, S. Gentry, and J. Popović. Example-based control of human motion. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004. 2.5

V. Inman, H. Ralston, and F. Todd. *Human Walking*. Wiliams & Witkins, Baltimore, 1981. 1.3.1

I. Jolliffe. *Principal Component Analysis*. Springer series in statistics. Springer-Verlag, 1986. 1.2, B.2, B.2.3

L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, 12(4), aug 1996. 2.6, 1.2

T. Kim, S. Park, and S. Shin. Rythmic-motion synthesis based on motion-beat analysis. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2003. 2.5, 2.19

H. Ko and N. Badler. Animating human locomotion with inverse dynamics. *IEEE Computer Graphics and Applications*, 16(2):50–58, mar 1996. 2.1.2

J. Korein. A Geometric Investigation of Reach. *MIT press*, 1985. 2.4.2, 2.2

L. Kovar and M. Gleicher. Flexible automatic motion blending with registration curves. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 214–224, 2003. 2.3.1, 2.11, 2.3.1, 2.5, 1.2, 1.6, 3.1, 3.3.2, 3.4

L. Kovar and M. Gleicher. Automated extraction and parameterization of motions in large data sets. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2004. 2.3.1

L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 473–482, 2002a. 2.4.1, 2.5, 2.17, 2.5, 1.3, 1.5.1, 3.1, 3.4

L. Kovar, J. Schreiner, and M. Gleicher. Footskate cleanup for motion capture editing. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 97–104, 2002b. 2.4.2, 2.1, 2.3

J. Kuffner. Goal-Directed Navigation for Animated Characters Using Real-Time Path Planning and Control. In *Proceedings of Workshop on Modelling and Motion Capture Techniques for Virtual Environments*, 1998. 2.6

R. Kulpa, F. Multon, and B. Arnaldi. Morphology-independent representation of motions for interactive human-like animations. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005. 2.4.2

T. Kwon and S. Shin. Motion modeling for on-line locomotion synthesis. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2005. 2.5

V. Labbé, O. Sigaud, and P. Codognet. Anticipation of Periodic Movements in Real Time 3D Environments. In *Proceedings of ABiALS Workshop*, 2004. 2.4.3

Y-C. Lai, S. Chenney, and S. Fan. Group motion graph. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005. 2.5

F. Lamiraux and J.-P. Laumond. From paths to trajectories for multi-body mobile robots. In *Proceedings of International Symposium on Experimental Robotics*, pages 237–245, jun 1997. 1.3.1

J. Laszlo, M. van de Panne, and E. Fiume. Limit cycle control and its application to the animation of balancing and walking. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 1996. 2.1.2

M. Lau and J. Kuffner. Behavior Planning for Character Animation. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2005. 2.6, 2.21, 1.1, 1.3.2, 1.6

B. Le Callennec and R. Boulic. Interactive motion deformation with prioritized constraints. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004. 2.4.2, 2.15

J. Lee and S. Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 39–48, aug 1999. 2.4.2, 2.15, 2.1, 2.4.1

J. Lee, J. Chai, P. Reitsma, J. Hodgins, and N. Pollard. Interactive control of avatars animated with human motion data. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2002. 2.4.1, 2.5, 2.5, 1.5.1, 2.1, 2.3.1, 3.1, 3.4

I. Lim and D. Thalmann. Construction of animation models out of captured data. In *Procedings of IEEE Conference Multimedia and Expo*, August 2002. 2.3.2

K. Liu and Z. Popović. Synthesis of complex dynamic character motion from simple animations. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 408–416, 2002. 2.1.3, 2.7, 2.4.1, 1.6

A. Maciejewski. Dealing with the ill-conditioned equations of motion for articulated figures. *IEEE Computer Graphics and Applications*, 10(3):63–71, may 1990. 2.4.2

J. MacQueen. Some methods for classification and analysis of multivariate observations. volume 1, pages 281–297, 1967. 2.3.2.1

Maya®. Alias systems corp. www.alias.com, 2005. 2.1.1

A. Menache. *Understanding Motion Capture for Computer Animation and Video Games*. Academic Press, San Diego, 2000. 2.1.3

S. Menardais, R. Kulpa, and B. Arnaldi. Synchronisation for dynamic blending of motions. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004. 2.4.1, 2.5, 2.16, 2.1, 2.3.1, 3.1, 3.2, 3.3.2, 3.4

T. Molet, Z. Huang, R. Boulic, and D. Thalmann. An animation interface designed for motion capture. In *Proceedings of Computer Animation and Social Agents*, pages 77–85, jul 1997. 2.4.2

J.-S. Monzani, P. Baerlocher, R. Boulic, and D. Thalmann. Using an intermediate skeleton and inverse kinematics for motion retargeting. In *Proceedings of Eurographics*, 2000. 2.4.2

MotionBuilder®. Alias systems corp. www.alias.com, 2005. 2.1.1, 1.1

MotionStar Wireless 2®. Motionstar. www.ascension-tech.com, 2005. 2.4

T. Mukai and S. Kuriyama. Geostatistical motion interpolation. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 1062–1070, 2005. 2.3.1

M. Müller, T. Röder, and M. Clausen. Efficient content-based retrieval of motion capture data. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2005. 2.3.1

F. Multon, L. France, M.-P. Cani-Gascuel, and G. Debunne. Computer animation of human walking: a survey. *The Journal of Visualization and Computer Animation*, 10(1):39–54, 1999. 2.1.2

M. Murray. Gait as a total pattern of movement. *American Journal of Physical Medicine*, 46 (1):290–333, 1967. 1.2

M. Neff and E. Fiume. Modeling tension and relaxation for computer animation. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 81–88, july 2002. 2.1.2

M. Neff and E. Fiume. Aesthetic edits for character animation. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 239–244, july 2003. 1.5

M. Neff and E. Fiume. AER: Aesthetic exploration and refinement for expressive character animation. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005. 1.5

K. Nipp and D. Stoffer. *Linear Algebra*. vdf Hochschulverlag AG an der EHTZ Zürich, 1992. B.2

C. O'Sullivan, J. Dingliana, T. Giang, and M. Kaiser. Evaluating the visual fidelity of physically based animations. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2003. 1.6

S. Park, H. Shin, and S. Shin. On-line locomotion generation based on motion blending. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2002a. 2.3.1, 2.11, 2.3.1, 2.5, 1.2, 2.3.2.1, 1.3, 1.6, 2.1, 2.2, 2.3, 3.4

S. Park, T. Kim, and S. Shin. On-line motion blending for real-time locomotion generation. Technical report, Computer Science Department, KAIST, 2003. 2.5

W. Park, D. Chaffin, and B. Martin. Modifying Motions for Avoiding Obstacles. *SAE Transaction*, 110(6):2250–2256, 2002b. 2.6

R. Paul. Robot manipulators: Mathematics, programming and control. *MIT press*, 1981. 2.4.2

X. Pennec and J.-P. Thirion. A framework for uncertainty and validation of 3d registration methods based on points and frames. *Int. Journal of Computer Vision*, 25(3):203–229, 1997. 2.3.2

K. Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, mar 1995. 2.1.3

J. Pettré, T. Siméon, and J.-P. Laumond. Planning human walk in virtual environments. In *Proceedings of International Conference on Intelligent Robots and Systems*, 2002. 2.3.1

J. Pettré, J.-P. Laumond, and T. Siméon. A 2-Stages Locomotion Planner for Digital Actors. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2003a. 2.6, 1.1

J. Pettré, J.-P. Laumond, and T. Siméon. 3D Collision Avoidance for Digital Actors Locomotion. In *Proceedings of International Conference on Intelligent Robots and Systems*, 2003b. 2.6, 1.3.1

N. Pollard. Simple machines for scaling human motion. In *Proceedings of Eurographics Workshop on Computer Animation and Simulation*, pages 3–11, 1999. 2.1.3

M. Ponder, G. Papagiannakis, T. Molet, N. Magnenat-Thalmann, and D. Thalmann. Vhd++ development framework: Towards extendible, component based vr/ar simulation engine featuring advanced virtual character technologies. In *Proceedings of Computer Graphics International*, pages 96–104, 2003. 2.3.2

Z. Popović and A. Witkin. Physically based motion transformation. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 11–20, 1999. 2.1.3, 2.7

W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press (second edition), 1992. 1.3, B.2

K. Pullen and C. Bregler. Motion capture assisted animation: Texturing and synthesis. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 501–508, 2002. 2.3.2

K. Pullen and C. Bregler. Animating by multi-level sampling. In *Proceedings of Computer Animation*, pages 36–42, 2000. 2.3.2

M. Raibert and J. Hodgins. Animation of Dynamic Legged Locomotion. In *Proceedings of Computer Graphics and Interactive Techniques*, pages 349–358, 1991. 2.1.2

P. Reitsma and N. Pollard. Evaluating motion graphs for character navigation. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 89–98, 2004. 2.5, 2.19

L. Ren, A. Patrick, A. Efros, J. Hodgins, and J. Rehg. A data-driven approach to quantify natural human motion. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2003. 1.6

C. Reynolds. Steering Behaviors For Autonomous Characters. In *Proceedings of Game Developers Conference*, pages 763–782, 1999. 2.4.3, 1.1, 1.3.1

C. Rose, B. Guenter, B. Bodenheimer, and M. Cohen. Efficient generation of motion transitions using spacetime constraints. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 147–154, 1996. 2.5, 2.3.2.1, 1.6

C. Rose, M. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–41, 1998. 2.3.1, 2.3.1, 2.10, 2.3.1, 2.3.2, 2.5, 2.5, 1.2, 2.3.2.1, 1.3, 2.2, 2.3, 3.1, 3.3.2, 3.4

C. Rose, P.-P. Sloan, and M. Cohen. Artist-directed inverse-kinematics using radial basis function interpolation. In *Proceedings of Eurographics*, 2001. 2.3.1

A. Safonova and J. Hodgins. Analyzing the physical correctness of interpolated human motion. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005. 1.6

A. Safonova, J. Hodgins, and N. Pollard. Synthesizing Physically Realistic Human Motion in Low-Dimensional, Behavior-Specific Spaces. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2004. 2.1.3, 2.3.2, 2.14, 1.6

M. Salvati, B. Le Callennec, and R. Boulic. A Generic Method for Geometric Contraints Detection. In *Proceedings of Eurographics, short presentation*, sep 2004. 2.4.1

A. Schache, P. Blanch, D. Rath, T. Wrigley, R. Starr, and K. Bennell. A comparison of overground and treadmill running for measuring the three-dimensional kinematics of the lumbo-pelvic-hip complex. *Clinical Biomechanics*, 16(8):667–680, oct 2001. 1.1

A. Schödl, R. Szeliski, D. Salesin, and I. Essa. Video textures. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 489–498, 2000. 2.5

H. Shin, J. Lee, S. Shin, and M. Gleicher. Computer puppetry: An importance-based approach. *ACM Transactions on Graphics*, 20(2):67–94, 2001. 2.3.1, 2.4.2, 2.1

H. Shin, L. Kovar, and M. Gleicher. Physical touch-up of human motions. In *Proceedings of Pacific Graphics*, oct 2003. 2.1.3, 2.8

K. Shoemake. Animating rotation with quaternion curves. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 245–254, july 1985. 2.2, 1.4, 3.3.3

P.-P. Sloan, C. Rose, and M. Cohen. Shape by example. In *Symposium on Interactive 3D Graphics*, pages 135–144, 2001. 2.3.1, 2.3.2.1

J. Starck, G. Miller, and A. Hilton. Video-based character animation. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005. 2.1.3

H. Stolze, J. Kuhtz, C. Mondwurf, A. Boczek, K. Johnk, G. Deuschl, and M. Illert. Gait analysis during treadmill and overground locomotion in children and adults. *Electroencephalography and clinical Neurophysiology*, 105(6):490–497, 1997. 2.4.2, 1.1

D. Sturman. Interactive keyframe animation of 3D articulated models. In *Course Note, SIGGRAPH Course Number 10, Computer Animation: 3D Motion Specification and Control*, pages 17–25, july 1985. 2.1.1

H. Sun and D. Metaxas. Automating Gait Generation. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2001. 2.1.2, 2.2, 2.3.1

M. Sung, L. Kovar, and M. Gleicher. Fast and accurate goal-directed motion synthesis for crowds. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2005. 2.6, 1.1

S. Tak, O.-Y. Song, and H.-S. Ko. Spacetime sweeping: An interactive dynamic constraints solver. In *Proceedings of Computer Animation and Social Agents*, pages 261–270, 2002. 2.1.3

L. Tanco and A. Hilton. Realistic synthesis of novel human movements from a database of motion capture examples. In *Workshop on Human Motion (HUMO'00)*, pages 137–142, dec 2000. 2.3.2

D. Tolani, A. Goswami, and N. Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models*, 62(5):353–388, 2000. 2.4.2

N. Troje. Decomposing biological motion: A framework for analysis and synthesis of human gait patterns. *Journal of Vision*, 5(2):371–387, 2002. 2.3.2

T. Tsumura, T. Yoshizuka, T. Nojirino, and T. Noma. T4: a motion-capture-based goal-directed realtime responsive locomotion engine. In *Proceedings of Computer Animation'01*, pages 52–60, 2001. 2.1.2, 2.6

M. Unuma, K. Anjyo, and R. Takeuchi. Fourier principles for emotion-based human figure. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 91–96, aug 1995. 2.1.3, 2.5, 2.3.1, 2.5

R. Urtasun and P. Fua. 3d human body tracking using deterministic temporal motion models. In *European Conference on Computer Vision*, may 2004a. 2.3.2

R. Urtasun and P. Fua. 3d tracking for gait characterization and recognition. In *Sixth IEEE International Conference on Automatic Face and Gesture Recognition*, pages 17–22, may 2004b. 2.3.2

R. Urtasun, P. Glardon, R. Boulic, D. Thalmann, and P. Fua. Style-based motion synthesis. *Computer Graphics Forum*, 23(4):799–812, 2004. 1.5.2, 1.2.1

M. van de Panne. From footprints to animation. *Computer Graphics Forum*, pages 211–223, 1997. 2.6

M. Veloso, P. Stone, and M. Bowling. Anticipation: A Key for Collaboration in a Team of Agents. In *Proceedings of Conference on Autonomous Agents*, 1998. 2.4.3

Vicon Motion System®. Vicon peak. www.vicon.com, 2005. 2.4, 1.1, 1.1

F. Wagner da Silva, L. Velho, J. Gomes, and S. Goldenstein. Motion cyclification by time × frequency warping. In *Proceedings of Brazilian Symposium on Computer Graphics and Image Processing*, 1999. 1.1

J. Wang and B. Bodenheimer. An evaluation of a cost metric for selecting transitions between motion segments. In *Proceedings of ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2003. 2.5, 1.5.1

J. Wang and B. Bodenheimer. Computing the duration of motion transitions: An empirical approach. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 335–344, 2004. 2.5, 2.5

C. Welman. Inverse kinematics and geometric constraints for articulated figure manipulation, 1993. 2.4.2

D. Wiley and J. Hahn. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Application*, 17(6):39–45, november 1997. 2.3.1

D. Winter. *Biomechanics and Motor Control of Human Movement*. Wiley, New York, 1990. 2.1.2

A. Witkin and M. Kass. Spacetime constraints. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 1988. 2.6, 2.1.3

A. Witkin and Z. Popović. Motion warping. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 105–108, aug 1995. 2.1.3, 2.4.1

W. Wooten and J. Hodgins. Simulating leaping, tumbling, landing and balancing humans. In *Proceedings of IEEE International Conference on Robotics and Automation*, apr 2000. 2.1.2, 2.5

W. Wooten and J. Hodgins. Animation of human diving. *Computer Graphics Forum*, 15(1): 3–13, March 1996. 2.1.2

K. Yamane and Y. Nakamura. Natural Motion Animation through Constraining and Deconstraining at Will. *IEEE Transactions on Visualization and Computer Graphics*, 9(3): 352–360, jul 2003. 2.4.2

K. Yamane, J. Kuffner, and J. Hodgins. Synthesizing animations of human manipulation tasks. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, 2004. 2.4.2

D. Zeltzer. Motor control techniques for figure animation. *IEEE Computer Graphics and Applications*, 2(9):53–59, 1982. 2.1.2

D. Zeltzer. Knowledge-based animation. In *ACM SIGGRAPH/SIGART, Workshop on Motion*, pages 187–192, april 1983. 2.1.2

J. Zhao and N. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics*, 13(4):313–336, oct 1994. 2.4.2

W. Zhao, R. Chellappa, and A. Krishnaswamy. Discriminant analysis of principal components for face recognition. In *3rd International Conference on Automatic Face and Gesture Recognition*, pages 336–341, 1998. 2.3.2

V. Zordan and J. Hodgins. Motion capture-driven simulations that hit and react. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 89–96, july 2002. 2.1.3, 2.3.1

V. Zordan and J. Hodgins. Tracking and modifying upper-body human motion data with dynamic simulation. In *Proceedings of Eurographics Workshop on Computer Animation and Simulation*, pages 13–22, 1999. 2.1.3

V. Zordan, A. Majkowska, B. Chiu, and M. Fast. Dynamic response for motion capture animation. In *Proceedings of ACM SIGGRAPH, Annual Conference Series*, pages 697–701, july 2005. 2.8, 2.1.3

Curriculum Vitae

Pascal Glardon
Born in Fribourg (Switzerland) in 1976

PROFESSIONAL ACTIVITIES

From 2002	**Research assistant, EPFL, Lausanne** Interest in character animation based on motion capture
2001	**Software engineer, Orell Füssli Printing, Zürich** Research and development of computer applications for the creation of security documents (banknotes, identity cards, stock shares).
2000 (4 months)	**Software engineer, Bitplane AG, Zürich** Contribution to the development of software for medical applications.
1999 (4 months)	**Software engineer (internship), SwissLife, Adliswil** Research and development of internet applications connected to databases.

EDUCATION

1995-2000	Engineering diploma in Computer Science, ETHZ, Zürich Title: ”Konsistente Triangulierung beim Schneiden in Tetraedergittern”.
1991-1995	Scientific baccalaureate (”*Maturité fédérale type C*”), Collège St-Michel, Fribourg.

LANGUAGES

French	Mother tongue
English	Read, spoken and written
German	Read, spoken and written
Swiss German	Spoken

LEISURE ACTIVITIES

Sport	Running, squash, hiking
Music	Keyboard, drums

SELECTED PUBLICATIONS

R. Boulic, P. Glardon and D. Thalmann, ”**From measurements to model: the walk engine**”, In *Proceedings of Optical 3D Measurement Techniques VI*, pages 167-174, sept 2003.

P. Glardon, R. Boulic and D. Thalmann, ”**PCA-based walking engine using motion capture data**”, In *Proceedings of Computer Graphics International*, pages 292-298, june 2004.

P. Glardon, R. Boulic and D. Thalmann, ”**A coherent locomotion engine extrapolating beyond experimental data**”, In *Proceedings of Computer Animation and Social Agent*, pages 73-83, july 2004.

D. Bielser, P. Glardon, M. Teschner and M. Gross, ”**A state machine for real-time cutting of tetrahedral meshes**”, In *Journal of Graphical Models*, 66(6), pages 308-417, nov 2004.

R. Urtasun, P. Glardon, R. Boulic, D. Thalmann and P. Fua, ”**Style-based motion synthesis**”, In *Computer Graphics Forum*, 23(4), pages 799-812, dec 2004.

P. Glardon, R. Boulic and D. Thalmann, ”**On-line adapted transition between locomotion and jump**”, In *Proceedings of Computer Graphics International*, pages 292-298, june 2005.

P. Glardon, R. Boulic and D. Thalmann, ”**Robust On-line Adaptive Footplant Detection and Enforcement for Locomotion**”, To appear In *The Visual Computer*, 2006.

P. Glardon, R. Boulic and D. Thalmann, ”**Dynamic obstacle clearing for real-time character animation**”, To appear In *The Visual Computer*, 2006.