Motion Adaptation Based on Character Shape

Lyard, Etienne

UNIVERSITÉ DE GENÈVE

FACULTÉ DES SCIENCES

Département d'informatique — Professeur José Rolim

FACULTÉ DES SCIENCES
ÉCONOMIQUES ET SOCIALES

Département de systèmes d'information — Professeur Nadia Magnenat-Thalmann

Motion Adaptation Based on Character Shape

THÈSE

présentée à la Faculté des sciences de l'Université de Genève
pour obtenir le grade de Docteur ès sciences, mention informatique

par

Etienne LYARD
de
Flaine (France)

Thèse N° 4020

GENÈVE
Atelier d'impression ReproMail
2008

UNIVERSITÉ
DE GENÈVE
FACULTÉ DES SCIENCES

Doctorat ès sciences
mention informatique

Thèse de *Monsieur Etienne LYARD*

intitulée :

"Motion Adaptation Based on Character Shape"

La Faculté des sciences, sur le préavis de Madame N. MAGNENAT-THALMANN, professeure titulaire et directrice de thèse (Faculté des sciences économiques et sociales – Département des systèmes d'information), Messieurs J. ROLIM, professeur ordinaire et co-directeur de thèse (Département d'informatique), B. TONDU, professeur (Institut National des Sciences Appliquées – Département de génie électrique et informatique – Toulouse, France) et R. BOULIC, docteur (Ecole Polytechnique Fédérale de Lausanne – Faculté informatique et communications – Institut des systèmes informatiques et multimédias – Laboratoire de réalité virtuelle – Lausanne, Suisse), autorise l'impression de la présente thèse, sans exprimer d'opinion sur les propositions qui y sont énoncées.

Genève, le 22 septembre 2008

Thèse - 4020-

Le Doyen, Jean-Marc TRISCONE

N.B.- La thèse doit porter la déclaration précédente et remplir les conditions énumérées dans les "Informations relatives aux thèses de doctorat à l'Université de Genève".

Nombre d'exemplaires à livrer par colis séparé à la Faculté : - 4 -

Acknowledgments

First of all, I would like to express my true gratitude to Prof. Nadia Magnenat-Thalmann who gave me the opportunity to work at MIRALab, taught me so many things and pushed me forward when I was losing my motivation. I would also like to thank all the MIRALabians for their friendship, discussions and support and especially Marlene Arevalo and Nedjma Cadi for their great models and animations.

I would also like to warmly thank the members of my jury, Prof. José Rolim (University of Geneva), Dr. Ronan Boulic (EPFL) and Prof. Bertrand Tondu (INSA Toulouse) for giving some of their time to read my dissertation and for their valuable and insightful commentaries regarding my work.

Last but not least, I would like to hug my mother, my brother Jérôme, Delphine, friends and family for their indefectible support and encouragements. My thoughts go to my late father, who showed me what dedication and perseverance really mean.

This research has been partly funded by the E.U. project Leapfrog IP FP6-NMP-515810.

Abstract

Motion is an important part of virtual environments. Indeed, virtual humans movements must be realistic to trigger the sense of immersiveness and producing such animations is not an easy task which requires hours of manual work by skilled animators. To overcome this issue, motion captured clips tend to replace traditional hand animation because they offer a very high level of realism with minimal manual work. These clips, however, have one major drawback, namely that they can be applied only to a body with similar shape and sizes as the person who was captured.

This dependence to the dimensions of the captured subject has inspired many research works over the past years, which aimed at making an existing clip applicable to differently sized humans, or adapt it to a particular surrounding environment. There exists various approaches which consider the *length* of the limbs to perform the adaptation, while only a few ones consider their *girth* to prevent the character from colliding with its environment.

In this thesis, we propose a method that can adapt the animation of a character according to its actual shape, thus preventing any self-collision to occur. This enables the creation of new applications, such as a Virtual Try On (VTO). A VTO enables a user to virtually try out garments. To achieve so, a virtual avatar must be deformed to match the user's sizes, and be animated so that the fitting can be estimated. This calls for an adaptation of the motion to make it free of self penetrations while keeping the natural look of the captured clip.

In this thesis, we present two adaptation methods, one based on inverse kinematics (IK) and one based on global optimization. The first one performs faster than the latter and is easier to implement, but the results are of higher quality using global optimization. Moreover, the global optimization approach enables to also control the balance of the character. We also present two novel algorithms for removing the foot skating. The first one is based on analytical inverse kinematics and the second one uses a closed form method to achieve the cleanup. The second one preserves the motion of the limbs unlike previous approaches which preserve the path of the character.

Eventually, the results are evaluated through the animation of a model for which we know in advance the weight of its limbs. These results are compared with the data

calculated through the algorithms presented here in order to validate them.

Résumé

L'animation de personnages est une composante indispensable des mondes virtuels. Il est cependant très difficile de créer des animations réalistes, et celles-ci doivent être adaptées à chaque personnage animé. Il existe de nombreuses techniques qui permettent de créer ces animations, et parmi elles la capture de mouvement est de plus en plus utilisée. Cette technique enregistre les mouvements d'une personne réelle, qui seront par la suite appliqués à un modèle virtuel. Le principal inconvénient est que les mouvements ne peuvent être efficacement appliqués qu'à un modèle qui ressemble à la vraie personne capturée. Dans le cas contraire, de nombreux artefacts apparaissent, tels que le glissement des pieds sur le sol et les pénétrations des membres dans le corps (auto-pénétrations).

Pour remédier à ces problèmes, diverses méthodes furent proposées par la communauté scientifique, de manière à pouvoir rapidement et efficacement adapter un mouvement au modèle auquel il doit être appliqué. Malheureusement, la plupart de ces méthodes ne considèrent que la longueur des membres pour réaliser l'adaptation. Les quelques méthodes qui prennent en compte la vraie forme du personnage ne sont applicables qu'aux pénétrations avec l'environnement extérieur ou aux auto-pénétrations des mains dans le corps.

Dans cette thèse, nous proposons de nouvelles méthodes d'adaptation, qui prennent en compte la forme du personnage animé. Ces méthodes sont un composant indispensable à la création d'applications utilisant des avatars déformables, par exemple les cabines d'essayage virtuelles, qui permettent à l'utilisateur de déformer un avatar pour qu'il/elle lui ressemble. D'autre part, adapter le mouvement à la forme propre de chaque personnage virtuel permet également d'obtenir des animations de foules avec une grande variété de mouvements même si le nombre d'animations de départ est faible.

Ces méthodes visent à corriger les glissements de pieds non désirés, les auto-pénétrations ainsi que l'équilibre du personnage. Deux méthodes ont été développées pour corriger les glissement de pieds: l'une fait appel à la cinématique inverse analytique, alors que l'autre se base sur une méthode analytique pour modifier la translation globale du modèle. Deux méthodes sont également proposées pour corriger l'animation du personnage: la première utilise la cinématique inverse, alors que la seconde optimise

l'animation de manière globale.

Correction du glissement des pieds

Le glissement des pieds d'un modèle virtuel est un phénomène certes déplaisant, mais qui se produit fréquemment. Celui-ci est causé par la façon dont sont représentées les animations de personnages: à la manière d'une marionnette, les personnages sont d'abord placés globalement dans la scène, et une fois ce placement effectué, la posture est adaptée pour le mettre dans la bonne position. Ainsi, si la taille des membres est changée, alors la translation globale ne correspond plus au mouvement des jambes, et on voit alors les pieds glisser sur le sol.

À chaque pas de temps de l'animation, nous commençons par extraire le pied qui est posé au sol. Pour cela nous utilisons une heuristique basée sur la comparaison entre le mouvement global du personnage et le mouvement de ses pieds: le point des pieds dont le mouvement correspond le plus au mouvement global du personnage est celui qui doit rester plaqué au sol.

La première méthode supprime le glissement des pieds en deux étapes. Premièrement la translation horizontale est corrigée, en calculant quelle est la translation qui fixe les points choisis au sol. Ensuite, la translation verticale est modifiée de manière à minimiser les pénétrations et/ou élévations des pieds par rapport au sol.

La seconde méthode opère différemment: le mouvement des jambes est modifié pour qu'il y ait toujours au moins un des deux pieds qui soit plaqué sur le sol. La hiérarchie des transformations du personnage est ensuite modifiée pour démarrer à partir du pied fixé au sol. Enfin, les corrections calculées sont alors propagées jusqu'au pas suivant pour obtenir un mouvement lisse et continu.

Adaptation des mouvements du personnage

Lorsque le diamètre des membres d'un personnage virtuel est modifié, on assiste à un phénomène d'auto-pénétration. Celui ci peut être soit créé par le mouvement exécuté par le personnage (par exemple placer ses mains sur ses hanches) ou bien par la forme du personnage elle-même (par exemple le bras peut pénétrer le torse simplement de part son trop gros diamètre). Ce deuxième type de pénétration est différent d'abord parce qu'il dure beaucoup plus longtemps (souvent toute la durée de l'animation) que le premier et qu'il n'y a que peu de joints sur lesquels on peut agir pour supprimer la pénétration.

Pour calculer les éventuelles pénétrations, nous commençons par assigner un cylindre par membre, de manière à calculer le diamètre moyen de chaque partie du corps. Ce modèle cylindrique nous est également utile pour estimer la distribution de la masse du personnage, indispensable pour la correction de l'équilibre. Le diamètre de chaque cylindre est estimé grâce à un processus similaire à une analyse en composantes principales: nous calculons la matrice de covariance de chaque nuage de points associé à chaque partie du corps, et le calcul des valeurs et vecteurs propres associés nous permet

d'en déduire un diamètre.

La première méthode d'adaptation modifie le mouvement de chaque membre individuellement puis corrige l'équilibre du personnage pour qu'il soit au moins aussi bon que celui du mouvement original. Les bras sont d'abord écartés du tronc, puis l'orientation des avant-bras est ramenée vers leur orientation initiale. Les jambes sont également écartées l'une de l'autre, et les hanches sont tournées pour faciliter la séparation des jambes. Une dernière étape de suppression des auto-collisions s'occupe des pénétrations crées par les mains, avant que l'équilibre ne soit corrigé.

La correction de l'équilibre se fait à travers la manipulation du Zéro Moment Point (ZMP) qui est l'équivalent dynamique du centre de gravité. Pour chaque pas de temps de l'animation originale, on calcule la distance entre ce point et le polygone de support du personnage. Après la modification de l'animation des membres, on modifie une dernière fois l'animation de manière à ramener ce point au moins aussi près du polygone de support que dans l'animation originale.

Plutôt que d'appliquer une valeur de correction par pas de temps de l'animation, nous utilisons des points de contrôles de courbes. Ceci a l'avantage de réduire la complexité de l'optimisation, mais également d'assurer que l'animation finale sera lisse, les points de contrôles étant eux même interpolés de manière lisse à travers l'animation.

La seconde méthode d'adaptation utilise la cinématique inverse analytique. Cette méthode est plus simple que la précédente, mais elle ne permet pas d'obtenir des animations d'aussi bonne qualité. L'orientation des avant-bras n'est pas corrigée, les hanches ne sont pas tournées pour faciliter la séparation des jambes, et l'équilibre du personnage n'est pas pris en compte. De plus, les corrections sont appliquées pour chaque pas de temps, et il est donc nécessaire de lisser les corrections après les avoir calculées.

L'adaptation d'un mouvement de haute qualité prenant plusieurs minutes, la première méthode n'est pas utilisable en temps réel. Pour palier à cela, une méthode d'interpolation de données pré-calculées a été développée. Celle-ci est basée sur les fonctions d'interpolation radiales, en utilisant la norme infinie et des échantillons choisis pour leur emplacement dans l'espace d'interpolation. Cette méthode nous a permis d'obtenir des résultats en temps réel, et donc d'utiliser les algorithmes décrits dans cette thèse pour des application telles qu'une cabine d'essayage virtuel.

Implementation et validation

Tous les algorithmes décrits dans cette thèse ont été implantés en C++, et intégrés dans l'environnement Open Scène Graph. Les modèles humains doivent correspondre au standard décrit par character studio de 3DS Max pour pouvoir retrouver l'humain virtuel dans la scène 3D. De plus, les algorithmes d'animation et de déformation des personnages étaient manquant dans OSG et ont donc été implémentés.

Pour valider l'algorithme de correction de l'équilibre et d'estimation de la distribution des masses, les données calculées ont été comparées à des données réelles. Pour

cela, la masse des différents membres d'un sujet a été capturée par absorptiomètrie, sa forme par un scanner 3D et ses gestes grâce à un système de capture de mouvement. Les trajectoires du ZMP calculées grâce aux données réelles ont été comparées avec les trajectoires obtenues en partant du modèle 3D uniquement. Les trajectoires obtenues étaient très proches, et après adaptation du mouvement la distance entre le ZMP et le polygone de support avait bien été ramené au niveau de l'animation originale.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Virtual characters are of great interest for the computer graphics community. Generate pictures that look human, move human and behave human is a challenging problem, stemming lots of different issues that must be solved depending on the application and context of the targeted application. Thus, many approaches were proposed already for addressing the technical difficulties that one encounters when following this path.
The directions of research dealing with this particular topic are many. Modeling, animation and rendering are the main aspects that were investigated, and as soon as a virtual human appears on a screen, this field of research becomes closely linked with other sciences such as cognitive, psychological and social sciences. This thesis, however, is only dealing with character animation, and more specifically with motion adaptation as it is a wide enough topic by itself, without considering aspects such as how an audience perceives the animation or the feeling that is triggered by such or such stimuli.

1.1 Character Animation

The general principle for animating a virtual 3D character - whatever it looks like - is as follows. First of all the character is given an underlying skeleton with a hierarchical structure (Figure 1.1) which is the object that drives the animation. Because of its hierarchical formulation (i.e. each limb is placed with respect to the limb onto which it is attached), all the joints rotations and segments offsets must be computed, along with one unique root transformation used for placing the skeleton in space. This approach can be seen as placing the character at its correct location and then adapt its pose for the current frame. It has many benefits, such as the possibility to easily edit the skeleton pose in order to make it match a desired configuration by simply rotating the joints (because of the hierarchical formulation, rotating the shoulder joint moves the entire arm), or the fact that a given animation can be applied on various skeletons without having to re-compute the rotation of all the joints. The drawbacks - tightly linked to the benefits - are that because there exist no explicit relationship between an animation and the sizes of a skeleton, a motion can be directly applied on only one skeleton and one must adapt - or retarget - the motion before applying it on another skeleton. This problem can be

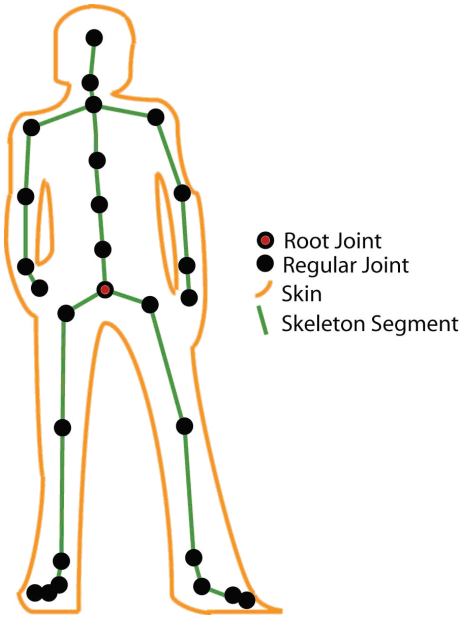

Figure 1.1: Principle of a character animation. The skeleton (in green) is first placed in the 3D space thanks to the transformation of a root joint. The skeleton is then put in the right pose by rotating its other joints, and finally the skin (in orange) is deformed according to the skeleton.

illustrated as follow: two characters with different sizes and shapes (for instance one tall and on small), while walking, do not travel the same distance when they perform the exact same number of steps. But because the global location of a skeleton is recorded once and for all when the animation is created, the global location of the root joint of the animation does not match the new length of the limbs, and the character has its feet sliding on the ground, if not penetrating it or floating in the air.

Once the underlying skeleton is animated, the character is given a virtual skin: this is the skinning stage (Figure 1.1). A virtual skin is a 3D mesh attached to the skeleton in such a way that it follows the skeleton animation. The attachment process consists of defining a relationship between each vertex of the skin and a subset of bones from the skeleton so that the skin follows the motion of the skeleton in a sound and realistic manner. Even if the skinning attachment is done very carefully by skilled animators, it can also yield to various unpleasant visual artifacts which are usually corrected afterwards by hand. Basically, the same adaptation issues that appear for the skeleton have their skin equivalent. For instance, if the skeleton animation was created with a slim morphology in mind, applying it to a fat character makes the body penetrate itself because the actual girth of each limb is now much larger..

Finally, once a nice looking skinned character is obtained, additional features are added: cloth, hair and more. However, because this third phase is out of the scope of this thesis, it will not be discussed here. A conceptual view of this animation process can be seen on figure 1.2.

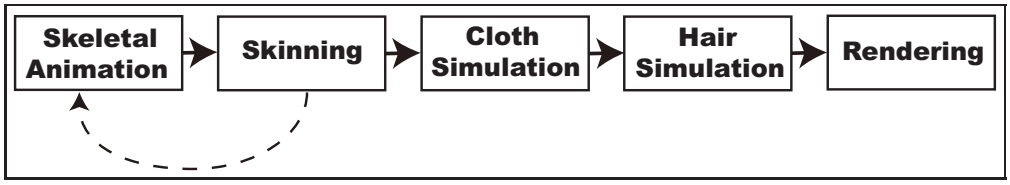

Figure 1.2: The character animation process. From an existing skeletal animation, a skin is attached to the skeleton bones. In case the skin conflict with the animation (because of self penetrations for instance) then the animation is modified (dashed arrow). Once a satisfying character animation is obtained, other elements can be added such as cloth simulation or hair. Eventually, the animation is rendered either in real-time or offline.

1.2 Motivation

MIRALab at the University of Geneva has a long history of innovations in the field of virtual human animation [MTK95, BCH+95, MHMTT97, CSMT98, SCPMT01, MTKM01, EMM04] for which motion captured clips [msw08, cw08] tend to be used more and more instead of previous animation techniques, which do not offer such a high level of realism. The acquisition of all these animation clips over the past few years now from a database of motion available for animating new models without having to perform new captures. However, as described in the previous section, each motion must be adapted to the target character before it produces a satisfying result. In the past, because computers were still slow and interactive applications not wide spread yet, each animation was prepared well in advanced by animators who adapted each motion individually so that it matched the target body, using all the off-line and on-line adaptation tools available, and correcting the remaining artifacts by hands. Nowadays, the emergence of high end graphics for the masses brought computer animation to a new level, and the customization offered to the end users has drastically grown. Today, in a variety of virtual reality applications, the user is able to customize his/her avatar so that it better represents him/her. Most of the time the user can only change the color of the 3D model along with various accessories (hat, glasses...), but recently appeared a class of application that enable to actually change the dimensions of the 3D avatar [SMT03, MTSC04] so that it not only looks like him/her, but it also has the same morphological characteristics.

For instance, MIRALab has an application called the *Virtual Try On* which allows a customer to try garments on-line without having to leave his/her home. For better try out and fit, a 3D avatar is deformed according to the customer sizes, and the clothes are then put on and animated with a standard catwalk. Such an application demands to adapt motion clips on the fly and with no user interaction. In case no care is taken when animating the newly deformed body, artifacts appear (figure 1.3) which prevent the customer from focusing on the cloth he/she is trying on. Previous approaches for motion adaptation do not take into account the skin (see chap. 2) and this is a real problem here as a big part of the tuning applied to the character deals with the skin shape and not only with the underlying skeleton. Hence even though previous approaches addressed numerous issues related to retargeting, a gap must be filled in order to create the missing components of a VTO application. Namely:

- A foot skating removal algorithm that leaves the original animation untouched.

Only the path of the character can be modified.

- A self-penetration removal algorithm that can address the penetrations induced by the character *shape*.
- A physically based balance correction that can operate with no user interaction.

Figure 1.3: Illustration of the self penetration issue. The animation was not correctly retargeted to the specific body it is being applied on, and one can see unpleasant artifacts taking place on the legs.

CHAPTER 2

Previous Works

2.1 Introduction

The CG community started early to investigate how to efficiently animate virtual characters by first applying well known techniques from the classical animation industry - mainly cartoons - to 3D characters [Las87]. At this early stage, because the existing guidelines pointed out how an animation should look like rather than how it should be done, all the animations were made by hand and high quality results required many hours of editing by well trained animators. Hence the community immediately started up to investigate how the animation process could be improved and which techniques could be applied to such topic. The mathematical tools that emerged first were direct kinematics [EW58, Lab66] (moving the shoulder joint of a character moves the entire arm) and key framing [BW71] (the artist defines key postures and the system interpolates the animation between them). Soon after came inverse kinematics [Pau82, Fea83] and its application to computer animation [GM85] which allowed one to directly manipulate the end effectors (hands and feet) for placing them in a suitable configuration. These tools only allowed animators to manipulate the character in a way that made the animation possible, and still it took many hours of work to animate a model. Thus researchers continued (figure 2.1) to look after improvements on how to make an animation, trying novel approaches for generating motion or reusing existing clips, with always the same goal in mind: make the animation pipeline easier and faster to use for the animator.

2.2 Motion Capture

It has always been difficult to animate 3D characters from scratch. In the early 60's, the first systems providing actual handles to control an animation were designed. In 1967, the first exoskeleton (called *data suit* at that time) was created by Lee Harrison III, an electronic engineer. Various competitive devices were created since then, such as magnetic trackers (first built by Polhemus) and optical systems. In the early 90's optical systems started to take over the animation business, thanks to the improvement

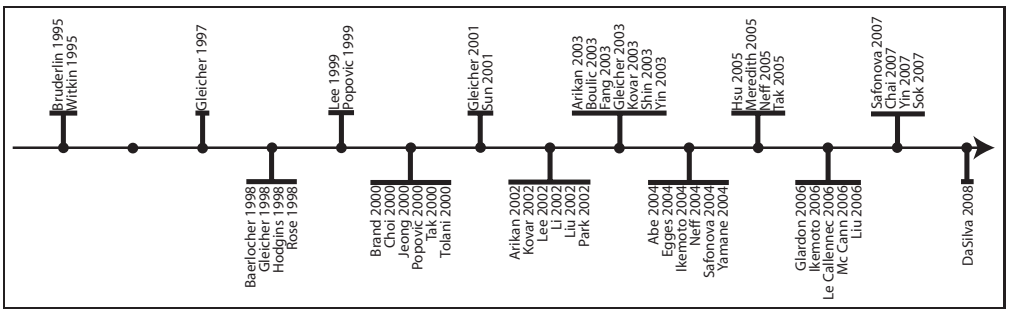

Figure 2.1: Timeline of the most significant works in motion adaptation.

of appropriate software tools which made the post processing easier. Instead of defining the poses of the character along the animation by hand, the motion is recorded on a real subject using markers placed on the subject's body.

Optical systems [msw08, cw08] are the most widely used for high quality animations of virtual character, probably because they are accurate and easily tunable to meet each capture requirements. This class of system uses infrared light coupled with spherical markers covered by reflective tape. Specially equipped cameras see the markers and reconstruct their location in 3D using simple triangulation principles, and the skeleton motion is then estimated from the markers [ZH03]. The main drawback of optical techniques is the inherent occlusion of the markers by the subject's limbs. This happens even if a high number of cameras are used and thus a post-processing stage is almost mandatory. Optical systems are thus primarily used for off-line applications, even if high end computers and improved post-processing techniques now allow to accurately estimate the character's pose in real time.
Exoskeleton systems [gmcs08] are less used than optical and magnetic trackers, probably because of their intrusive nature. The principle is the following: the subject to be captured wears an exoskeleton onto which the joint angles are retrieved, thus directly delivering the data necessary for animating the skeleton.
Magnetic trackers [tw08, web08a] are less accurate than the two other classes, but they have the advantage to provide the orientation of a marker along with its 3D position in space. Fewer markers are required to evaluate the configuration of a body, and they are often used in real time application as they demand less calibration compared to optical trackers. They also are insensitive to occlusions, which makes them well suited to real-time and interactive applications. The drawback is that using magnetic fields, these systems must operate in an environment free of metallic parts.

The massive use of motion captured data over the past twenty years improved so drastically the realism of animations that this approach became the starting point of almost all the researches conducted afterward. Even though it remains challenging to neatly capture a motion clip, tools were developed by the hardware manufacturers for efficiently post-processing the raw data outputted by their systems [pptfVP08, pptfMA08] but because this is somewhat out of the scope of this document, it will not be discussed here. Rather, we start from the point where a nice looking motion clip was captured, and the animation data is converted to a usable format (joint angles, global frames...) for later use by the animator. Even then, it still takes a lot of man-

Figure 2.2: Two tracking strategies for motion capturing human movements. From left: optical (Vicon [msw08]), right: magnetic (Ascension [tw08]).

ual work to map the motion onto the character and if no special care is taken on how the operation is performed, plenty of artifacts may appear. Thus, the research community focused their efforts in developing tools and methods for reducing the amount of work required to obtain the desired final animation without degrading quality. Because many aspects must be considered when dealing with such process, various directions were investigated with different objectives in mind. Four main categories (figure 2.3) can be distinguished: Motion Editing, character based adaptation, environment based adaptation and motion blending (figure 2.4).

- Motion Editing deals with the works which tried to make the manual touchup of a motion easy and intuitive.
- Character based adaptation aims at making the motion of a character look more natural by re-establishing constraints which were discarded by the animation pipeline, would they be physical properties or other criterions.
- Environment based adaptation gathers works which tried to make the character motion fit into a surrounding environment. For instance grasp an object in the 3D world, reach a handle and more.
- Motion Blending is not clearly related to retargeting, but it was included in this section anyway as many techniques employed there are close from adaptation techniques.

Two extra categories can be added to the above four: foot skate cleanup and motion synthesis. The first one aims at cleaning the motion from foot skating artifacts, while the second aims at re-generating motion from scratch. Even though they are not directly

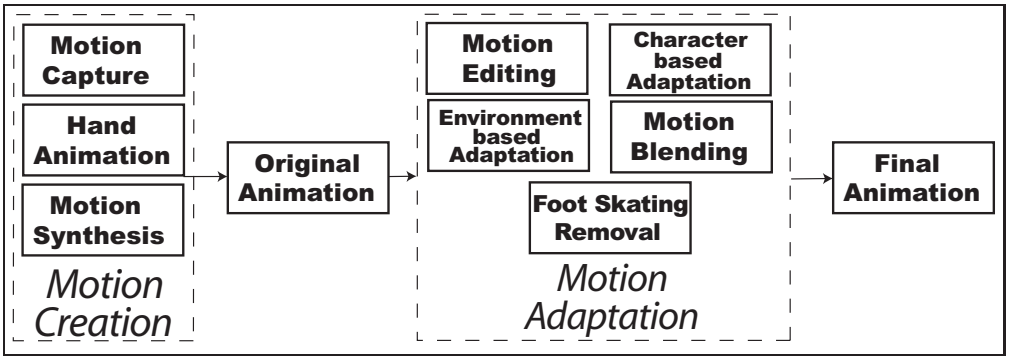

Figure 2.3: An overview of the motion creation process. First the initial skeletal animation clip is created. For this, various techniques can be employed such as motion capture, hand animation or motion synthesis through dynamic simulation. Once a first animation clip is obtained, it is adapted so that it better match the animator requirements. The changes can be driven by the animator through motion editing, adapted to the specificities of the character being animated or to the environment into which the character evolves. Using motion blending, several clips can be mixed to produce a new one and eventually the foot skating is removed to obtain the final animation clip.

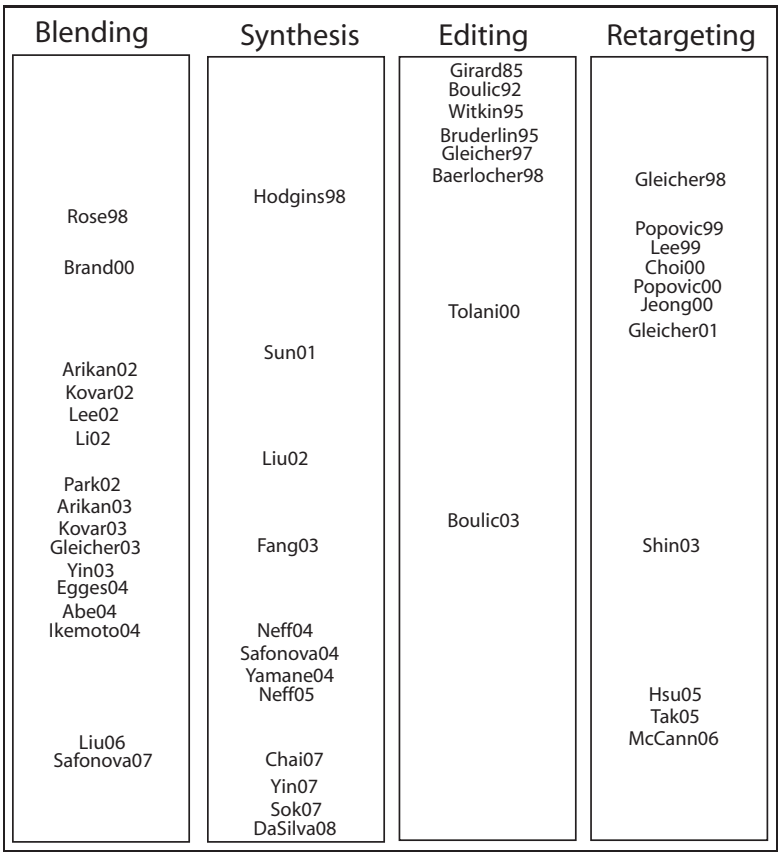

Figure 2.4: Main goal of various contributions in the field of motion adaptation. Several trends can be drawn from this mapping: in the years 1999 to 2001, the focus was clearly on pure motion retargeting. Later on motion blending and then synthesis seem to have attracted the attention of the community.

related to motion adaptation, many of their concepts and approaches are relevant here. Moreover, several recent works used motion synthesized from examples in order to generate motion clips rather than adapt an existing one. Compared to previous approaches, it has the advantage that because the motions are synthesized, they accurately fit the production needs. The drawbacks are that a synthesized motion, even if of high quality, never completely reaches the level of realism provided by motion captured clips.

We will sometimes make the distinction between character based and environment based adaptation, but we will often group these two categories for the sake of simplicity. Moreover, all the proposed approaches aim at modifying an existing animation and can be applied to more than one category. Most of them employ a combination of numerical techniques to achieve their goal, thus it would not make sense to group them per class of technique. Thus, the related work will be reviewed somewhat chronologically rather than per category, with a grouping by trends, as seen on figure 2.5.

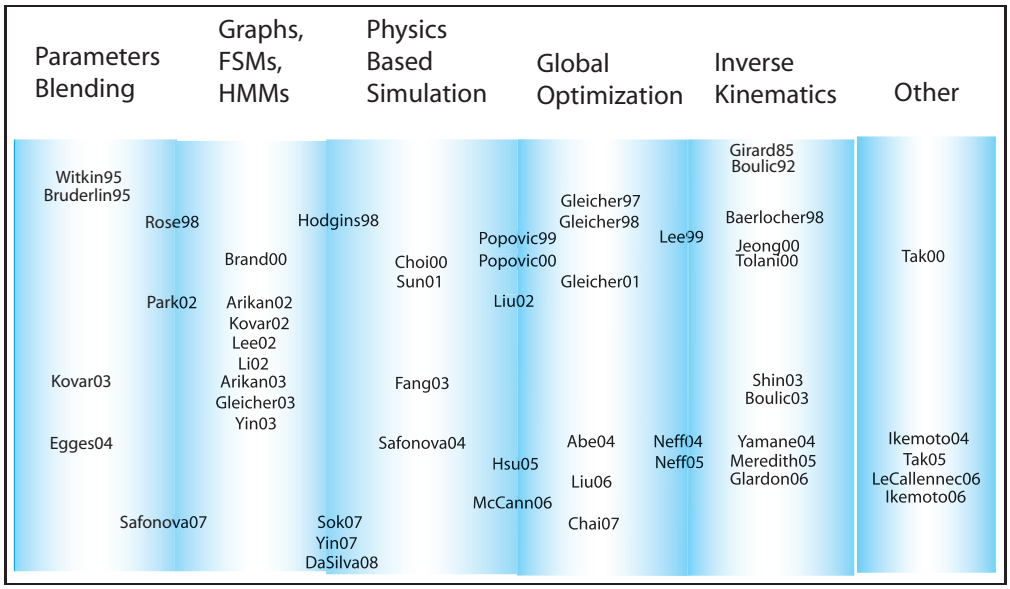

Figure 2.5: Mapping of various works over the main numerical technique they employ. Parameter blending includes all the interpolations done on the parameters driving the joints angles. Graphs, Finite State Machines (FSM) and Hidden Markov Models (HMM) denote the use of a statistical process. Physics based simulation speaks for itself and so does Global Optimization. Inverse Kinematics includes all the processes which aim at driving an effector towards its goal (would it be numerical or analytical IK), and *other* gathers the works which could not be fit in any of the above mentioned categories, like Kalmann filtering or limbs transplant for instance. No clear trend can be drawn from this picture even though graphs were investigated much during the years 2002 and 2003.

2.3 Early Approaches

When motion capture techniques entered the CG scene, it became obvious that most captured clips could not be used as is: they should be modified after the capture so

that they match the animation requirements. Indeed, even if the capture sessions are well organized, it is almost impossible to precisely know beforehand how the motions should be performed: the virtual environment is not here to guide the actors, neither are the other virtual characters with whom the actor must interact. Thus, the community soon started to investigate how existing clips can be modified in a sound way so that all the interesting features of the original motion are preserved.

This is not a trivial issue because the animation parameters do not intuitively map to the motion performed by the character. Witkin and Popović [WP95] were among the first to propose a way to intuitively modify such parameters. Their approach directly deals with the parameter curves (i.e. the value of joint angles over the animation) by blending the original curves with user specified values. They managed to make it intuitive by allowing the user to specify key frames over the animation. Using inverse kinematics, the limbs of the character can be easily moved to the desired posture which then provides parameter values to the interpolation system. Once key frames are given by the user, the system simply interpolates the original curves with these new values using cardinal splines. This approach makes it possible to tweak existing motions by specifying key postures by hand. Even though this already was an improvement compared to the no-interaction status that existed previously, this approach did not solve the problem entirely. Indeed, even though the interpolation drives the character to the target pose, it does not control how this is done, and more specifically, it may add high frequency features to the motion curves, yielding to non realistic artifacts in the final motion.

Another early work on the subject is from Bruderlin and Williams [BW95]. Their motion signal processing technique treat the trajectories of each DoF of the skeleton as a time varying signal to which they apply a wavelet decomposition. This allows for the tuning of individual levels of the decomposition, thus resulting in smooth and global changes applied to the entire motion.
Moreover, individual levels of several motions can be blended together, thus resulting in a newly blended animation. Because the blended motions must be matched to each other, the motions are time warped before they are blended. This warping is done using an adaptation of the Sederberg method [SG92] for shape matching. Eventually, a wave shaping concept is introduced. It says that it is possible to change a motion by applying a shape function which is multiplied to the values of a particular joint angle. However, as this approach directly deals with the parameter curves (just like [WP95]), it is somewhat difficult to use because of the non intuitive nature of these curves.

2.3.1 Global Optimization

Michael Gleicher [Gle97] addressed this weakness in his cornerstone paper ”Motion editing with spacetime constraints”. As for Witkin, he proposed to edit the target character pose by hand, but his method is completely different regarding how the adaptation is performed. Gleicher formulated the editing as a constrained spacetime optimization problem [WK88]: the parameter curves of the motion are given as an input to the system, and spatial constraints - the target pose specified by the user - are added automatically from the user high-level input. Once the problem is setup this way, the global

optimization algorithm matches spline curves on top of the motion curves so that the spatial constraints are met. The magnitude of the curve is to be minimized so that the resulting motion remains as close as possible from the original clip, and the distance in frame between the spline control points determines to which extent high frequency features can be added or not.

The next year, Gleicher again extended his work to apply it to motion adaptation [Gle98, GL98]. Instead of specifying target poses, the user only gives the constraints that must be preserved by the adaptation (e.g. foot plants, grasp...). The system then uses the same strategy as for Gleicher's previous work, namely re-establish the constraints after applying the motion to a character with other dimensions. The spacing between the control points is calculated automatically by first launching an optimization with a sparse set of control points, and re-launches the algorithm with a denser set of points in case the constraints could not be enforced during the first optimization.

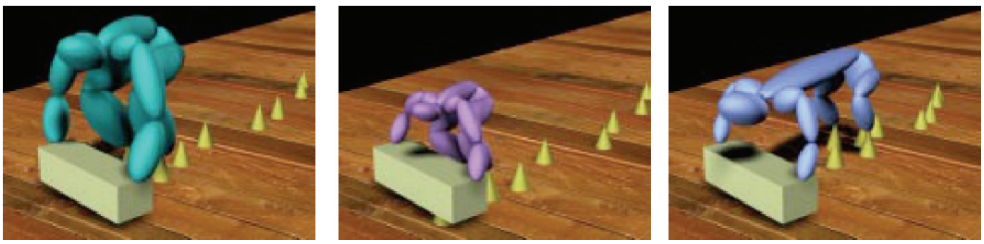

Figure 2.6: A picking motion clip (left) is retargeted by Gleicher [Gle98] to match two new characters (center and right). Image from [Gle98] used by permission.

2.3.2 Motion Creation

Approximately at the same time, the very first works which aimed at synthesizing human motion from scratch appeared. Hodgins [Hod98] used finite state machines to drive physical simulations in order to generate human athletes' performances: running, cycling and vaulting (figure 2.7). This approach made use of many assumptions in order to constrain the dynamic system and hence reduce the number of variables to be controlled. Even though promising results were obtained, the generated motions were still rigid and robot-like looking, probably because of the symmetry and canonic behavior of the avatars. This approach was exploited a few years later by Sun et al. [SM01] who proposed a more generic gait generator system using three layers of abstraction in order to generate walking motion of human figures. The system first generates generic walk movements, which are then adapted using the second module to the specificities of the desired animation (irregular terrain, curved path...). The third module modifies with a high level controller the parameters that must be fed into the two first modules, giving the animator a better control on the results which are still far from approaching the quality and realism of motion captured clips.

In parallel, the idea of blending motions together to generate new, better matched motion started to be investigated. Rose et al. [RCB98] formalized the manipulation of motion clips by introducing the notion of verbs (i.e. what is to be done) and adverbs

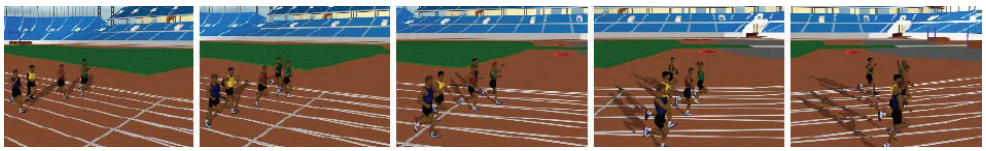

Figure 2.7: A virtual race created by Hodgins [Hod98]. Image from [Hod98] used by permission.

(i.e. how this should be done). Once tagged, the clips can be combined in order to produce new motions. However, their approach looks difficult and time consuming to use because of the following. First the set of example motions must contain only similar motions. For instance, a set of example walks must have clips which start on the same foot, have the same number of steps and not contain side motions such as a hand wave for instance. A pre-processing stage requiring manual labeling is perform at first in order to construct a Verb Graph, i.e. pieces of motion (verbs) along with the possible transitions between them. Each motion clip is re-parameterized to a generic time so that a generic time value t yields to similar configuration over the clips. Additionally, each motion clip is placed somewhere in an abstract Adverb space which somehow illustrate the high level features of the motion (happy, sad...)
Eventually, the motion curves themselves are represented using spline curves, which then enable one to interpolate control points instead of actual values of the parameter curves. Once the verb graph and adverb mapping is constructed, a radial basis function is assigned for each verb, and the system is now able to blend motions in order to generate new ones which correspond to a specific location in the abstract adverb space. This verbs and adverbs idea was used and extended by Park et al. [PSS02]. They changed several aspects which made the motions of higher quality and easier to control. They used incremental mapping in order to prevent the time to be reversed during the alignment of the motions [BMTT90]. They used cardinal basis functions instead of radial basis ones so that the blending produces better looking animations. They also improved several other details, such as the use of a logarithmic map during the blending, but the overall philosophy of their approach remained similar to the one from Rose et al.

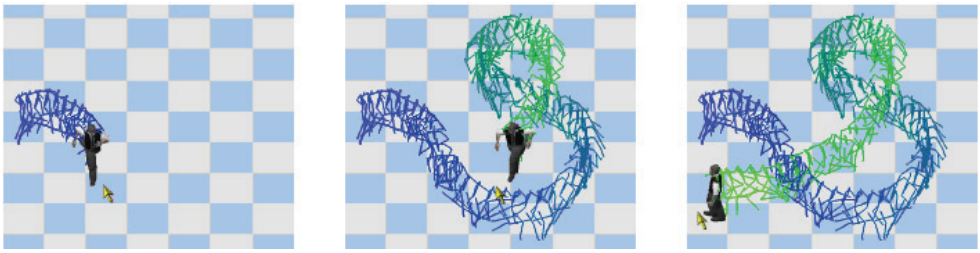

Figure 2.8: A virtual character chasing the mouse pointer. Its locomotion was generated by Park et Al. [PSS02]. Image from [PSS02] used by permission.

2.4 Higher Level Interactions

Among the various strategies that were tried out in these early days, optimization and inverse kinematics appeared to be among the most powerful tools to act on motions. The associated algorithms remained tracktable, but to interact with them, a designer had to understand the complex mecanisms that made them work. Thus, to simplify the control over an animation, higher level approaches were developped.

2.4.1 Physics of Motion

Popović and witkin [PW99, Pop00] first tackled the physical aspects of human motions. Their goal was to change the context of a motion, for instance make a walk sequence go uphill. Their method enforces the physical criteria on a simplified version of the character so that the optimization has a chance to converge to the desired solution. Here is an outline: first the character is simplified. Then a spacetime simulation is fitted onto this simplified animation so that it can be used as a starting point for the edit. The constraints imposed by the user can now be changed in order to match new needs (e.g. go uphill), and they are then re-enforced by re-optimizing the spacetime simulation. The resulting simplified animation is then mapped back to the character's original skeleton using a minimum displacement mass criterion.
This first work on the topic produced convincing results, but it suffers from the use of the simplified skeleton, which requires the user to specify by hand how the original model should be reduced. Moreover, because the retargeting is performed on the simplified model only, fine effects such as gait adaptation or collision avoidance simply cannot be dealt with by this approach.

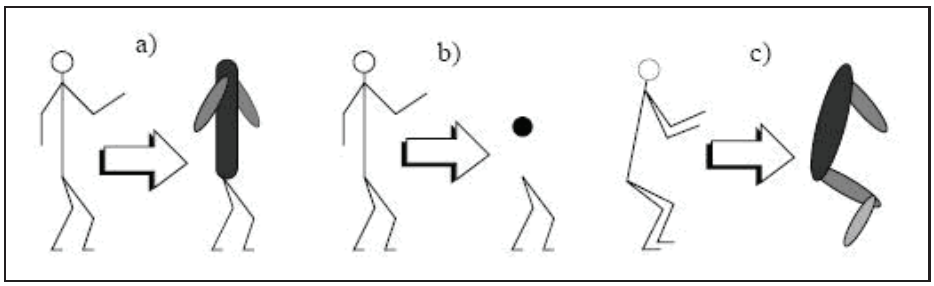

Figure 2.9: Some of the simplifications of the skeleton structure which are performed by Popović and Witkin [PW99]. a) Elbows and spine are abstracted away, b) upper body reduced to the center of mass, c) symmetric movement abstraction. Image from [PW99] used by permission.

Liu and Popović [LP02] describe a framework able to generate complex and physically correct motions from simple motion sketches drawn by the animator. The user simply has to define a few key frames along the animation in order to give the system a hint about what is to be performed. Then the system estimates - again thanks to physics principles coupled with global optimization - what would be the physical simulation that goes through the key frames previously defined. For all kind of constrained optimization involving physics, the bottleneck of the computation lies within the computation of the first derivative of the torques, which is quadratic with respect to

the number of degrees of freedom that the model has. Fang et al. [FP03] addressed this problem by proposing a new way of computing these derivatives in linear time. This enabled them to run the same kind of simulations as in [LP02] with up to twenty-two degrees of freedom, at interactive rate.

Abe et al. [ALP06] used once again a global optimization approach in order to modify highly dynamic motions. The system uses a momentum based parameterization for the global optimization, along with a spline based interpolation in order to reuse existing motion. Their system not only retargets clips, but is also able to interpolate existing clips to generate new ones which did not exist before.

Liu et al. [LHP06] composed the motion of several characters, thus considering the environment as another moving character. They reused the idea of space time optimization, but because such algorithm can be hard to make converge even for a single character, they slightly modified the algorithm performing the optimization. Time warping was added to the classical constraint based motion adaptation so that the timing of several motions can be tweaked to make them fit together. Both the timing and the constraints timing can evolve during the optimization process and dynamic constraints are used to ensure the physical plausibility of the resulting motions. Eventually, a block coordinate descent strategy is used to make the system converge. Basically, the block coordinate strategy fixes a subset of the problem's variables while the remaining *free* variables are optimized. The sets of fixed and evolving variables are alternated during the optimization and a final optimal state is thus reached as if all the variables would have been considered at the same time.

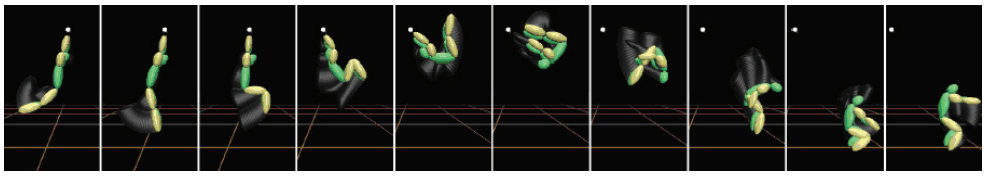

Figure 2.10: An animation produced by Fang and Pollard [FP03]. Image from [FP03] used by permission.

Tak et al. [TSK00] proposed to take into account the Zero Momentum Point (ZMP) in the retargeting process (c.f. section 4.7). For human motions, this point must lie within the supporting area (the foot sole) while the character is standing. They setup a classical spacetime optimization framework as the one described earlier, and added an extra constraint: make the ZMP lie within the supporting area. This work is not meant to retarget motions from character to character, but rather to make an existing motion clip finely tuned to the character onto which it is applied.
Tak and Ko [TK05] further refined their work and dropped the global optimization from their approach. Instead they used a Kalman filter [Sim06]. Kalman filtering is a widely used technique when it becomes necessary to estimate the actual state of a system from noisy observations. The filter includes a model of what can and cannot happen, and when fed with observation measures, it has the ability to give an optimal estimation of the real state of the system. It can appear bizarre to use such a filter in order to modify an already existing phenomenon, but this approach has proven successful. The real measurements are taken from the original motion clip, and the filter acceptable states

are defined by example motions which are known to be correct, this data set being extended with the kinematics and dynamic constraints defined by the user.
The process of retargeting is then simply to make the real measurement go through the filter, and retrieve the retargeted motion as a filter output. Compared to previous approaches which used global optimization, this filter technique is much faster, but it remains unclear whether the gain is due to the change of philosophy or rather to the fact that only one frame at a time is taken into account. Also, Kalman filtering can be hard to setup, especially regarding the confidence put in both the measurement and filter model. These two must be balanced by values given by the user (i.e. a measurement and model noise), and wrong values can dramatically degrade the quality of the final results.

McCann et al. [MPS06] tackled the issue of retiming animations in order to adapt them according to high level criterions such as the weight of the limbs. They first estimate the contact forces and torque exerted by the character joints. They then introduce a retiming factor p which they re-inject into the equations of motion in order to make them depend on p (which allows the retiming itself). Then they calculate the maximum physically plausible values for the first and second derivatives of p for a given min and max torque values applicable by a specific joint (i.e. how strong is the character), which gives a polygon within the derivatives of p must lie in order to keep the motion physically plausible. This way, it is possible to know which retiming can or cannot be accepted.
Having this feasibility region available, a user can then impose features to be achieved by the retiming, such as the look of a motion (effortless, stressed, heavy...). These high level parameters are then transformed into parameters of the simulation such as limb weight or target torques. Another function is then optimized taking into account the retiming objective and also the look objective.

2.4.2 Hybrid Approaches

Lee and Shin [LS99] were the first to propose a hybrid approach for editing a motion clip. Rather than using a spacetime optimization algorithm, which is slow and difficult to setup compared to analytic approaches, they developed a hybrid numerical/analytical solution. With the key-framed poses imposed by the designer, they begin by retargeting the root joint coarsely - basically they simply average the displacement imposed by the constraints taken separately - which more or less puts the character where it should be. Then again a constrained optimization solver finds the configuration of the torso that best matches the constraints, and eventually the limbs are adapted using an analytic inverse kinematics method [TGB00]. Compared to the previous approaches, this work has the benefits that it performs faster and is easier to setup. However, the quantitative performance of both approaches remains unclear: it is possible that a solution to the problem imposed by the user existed but because some possibilities were discarded by the multi-step approach of Lee and Shin, then the system might not be able to find it.

Choi and Ko [CK00] reused the hybrid philosophy, and achieved real-time performances for their system. They used inverse kinematics and inverse control rate in order to perform the retarget. The user defines target positions for the end effectors of the character, which are then reached by adapting the character pose. Unlike regular

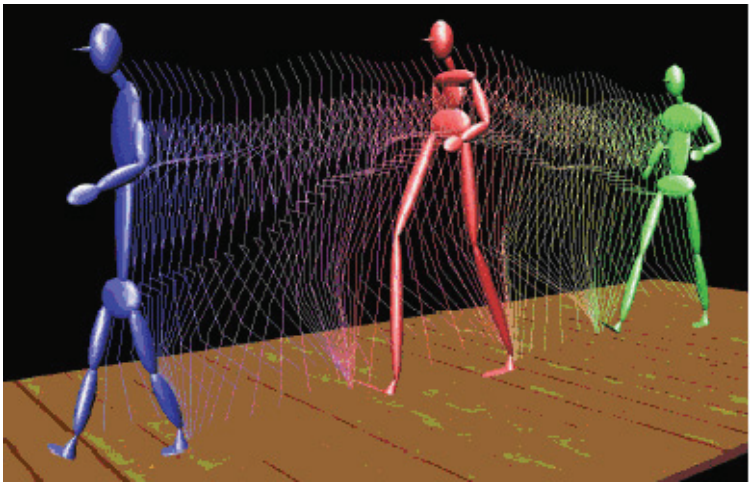

Figure 2.11: A character is resized on the fly, its motion being adapted accordingly by Lee and Shin [LS99]. Image from [LS99] used by permission.

inverse kinematics, inverse control rate estimates joints velocity instead of their configuration. The calculated velocity is then integrated over a time step in order to get the real character configuration. The primary goal of the inverse kinematics system is of course to comply with the constraints imposed by the user, and a secondary goal - to be achieved using only the remaining degrees of freedom - is defined in order to make the resulting motion looks like the original one as much as possible. Estimating the velocity of the joints rather than their configuration has another advantage: there is no need to smooth the resulting motion as it was already done by the estimation/correction scheme of the inverse control rate.

In 2001 Gleicher [Gle01] proposed a method which modifies the characters path instead of their pose. He reused a lot of his previous work for adapting the character's pose, and extended it so that it becomes possible to simply modify which path is followed by the character. For doing so, he first defined the path to follow as a time-varying vector value $P(t)$, which gives the location of the character at any time t. He also defined the orientation $R(t)$ of the character so that it is possible to control where it faces during the walk (e.g. walk sideways). The system estimate the initial path and orientation $P_0(t)$ and $R_0(t)$ while the user specifies the target quantities $P(t)$ and $R(t)$. With this as an input, the character is first placed at the right location for each frame, and its pose is adapted afterwards according to user-defined constraints such as foot plants.

Following the trend given by Lee and Choi, Shin et al. [SKG03] proposed to physically repair motion clips using closed form methods and cookbook-like procedures. Instead of solving a big nonlinear system as for other approaches, they first define which joints can be modified and how. Then they calculate the ZMP and adapt the body posture by simply rotating the hip joint. For flying phases (i.e. when the body is not in contact with the ground), the momentum is assumed constant and the method enforces the center of gravity of the body to follow a hyperbolic trajectory. As the process is iterative and per frame, each frame correction is smoothed along the animation so that no big acceleration of the joint angles is created. Even though ad-hoc compared to other

approaches, this method is able to produce convincing results at low cost.

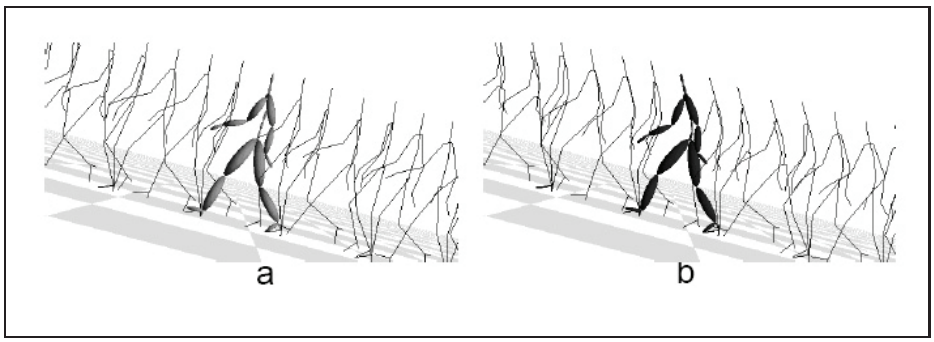

Figure 2.12: An adaptation performed by Shin et al. [SKG03] for an uphill walk. On the left hand side is the original motion, and on the right is the corrected one. Image from [SKG03] used by permission.

At this point, there exists a few approaches that allowed modifying a motion according to the user needs while preserving user-defined constraints such as foot plants or objects avoidance. However, all these approaches assumed that the constraints imposed to the character are indeed solvable and that an optimal solution satisfying all the constraints exists. What if this is not true? What if the character has to grasp at the same time two different objects which are too far apart for being reached simultaneously? This question was answered by Baerlocher and Boulic who introduced the notion of priority among constraints [BB04, BB98a]. Inverse Kinematics (IK) relies mostly on Jacobian matrices, i.e. the partial derivative matrix of a phenomenon, in order to estimate what to do with an articulated chain so that its end effector reaches a specific target. The method they proposed first satisfies the highest priority constraints, and subsequently builds updated jacobian matrices so that the upcoming pose modifications keep preserving the already satisfied constraints. They were thus able to define an arbitrary number of constraints, each with a specific level of priority, unlike previous approaches which allowed only two levels of priority [BT92]. Explained in more understandable terms, the principle of their method is intuitive: once a constraint is satisfied, the next jacobian matrix is updated so that the degrees of freedom which might break the previously satisfied constraints are removed. For instance, if a foot must be planted on the ground, then the next matrix allows the leg to rotate around the vertical axis, but a vertical motion of the foot itself is prevented by zeroing the related jacobian matrix coefficients.
This principle of Prioritize Inverse Kinematics was later used by Boulic [BLHB03] and Le Callennec [LB06a] for editing motion clips. Even though PIK performs well even with complex sets of constraints, it does not have frame to frame continuity enforcement; hence discontinuities in the character pose are to be expected.

Yin and Pai [YP03] designed a system which uses foot pressure in order to estimate what is being done by a user, select the corresponding motion segments from a database and blend them together in order to produce a new motion. A ten components feature vector is extracted from the foot pressure being recorded and these vectors are then compared over a time frame with previously recorded data using the Mahalanobis

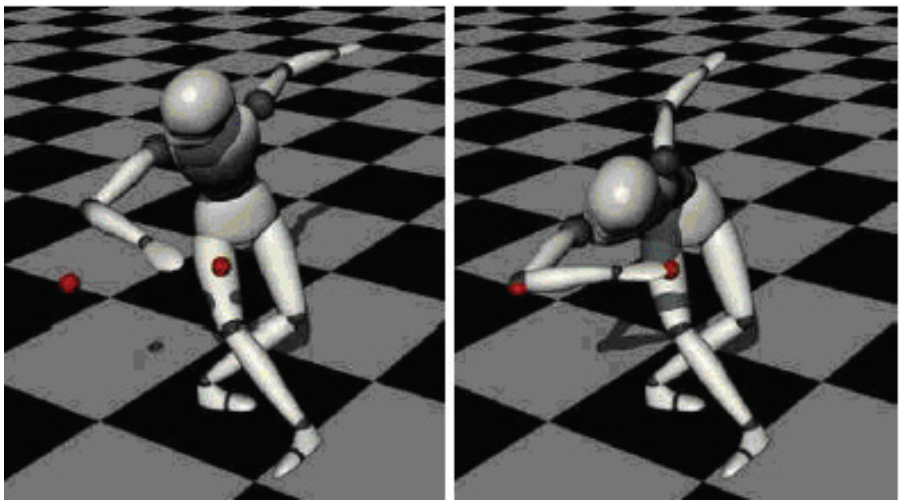

Figure 2.13: Kinematic modification of the skeleton pose performed by Boulic et al. [BLHB03]. Image from [BLHB03] used by permission.

distance.
Standard motion graph techniques is then applied in order to blend the motion sequences, and time warping is also applied in case the system detects that the motion is being performed at a different rate. Eventually, a closed form inverse kinematics algorithm is also proposed by the authors in order to enforce kinematics constraints such as foot planting. Their system is able to efficiently track and re-synthesize actions being performed by a user with only one second delay, and this by taking only the foot pressure data as an input.

2.4.3 Collision Avoidance

Preventing collisions or penetration with the environment and/or with the character itself has also been investigated by the community. Applying a motion to a body that it does not match produces interpenetration between objects in case of a virtual environment, while in the case of real robots it might turn out to be much more problematic, as it can yield to damaged equipment.
Zhao and Badler [ZB94] were among the first to address this issue. They introduced the concept of *sensors* attached to the body which monitor the collisions occurrence. For a set of collision primitive (ellipsoids, half-spaces and cylindrical tubes) they defined potential functions, which are positive inside the collision volumes only. In case a positive potential is detected, an IK step takes place to minimize the potential of the *sensor*.

This concept of *sensor* was later integrated in a framework dedicated to virtual humans by Boulic et al. [BHT97]. He and others later improved this approach by coupling it with prioritized IK. Instead of using potential fields, they defined a damping volume around the collision primitives [PBCM05], which smoothly prevents the *observers* (an entity analog to a *sensor*) from colliding with the surrounding environment or itself [PMM+07]. Once coupled with a postural control engine [PMRB06], which animates the character from mocap data, it is thus possible to control an Avatar in real-time with

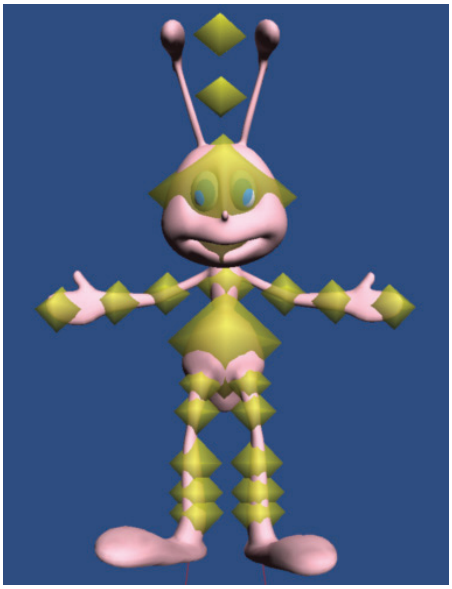

Figure 2.14: Collision volumes used by Jeong and Lee [JL00]. The character itself is colored light pink and the collision volumes are semi-transparent yellow. Image from [JL00] used by permission.

collision avoidance.
Kuffner et al. [JNK+02] proposed to use sets of threshold distance between specific points on the body to monitor and prevent self collision. As in the previously mentioned approaches, the potential collisions are detected before they actually happen, and are thus prevented rather than corrected. Jeong et al. [JL00] correct rather than prevent such artifacts, in case the animation is applied to a character slightly different from the capture subject. They also use collision volumes attached to the skeleton so that the character shape is taken into account and correct the penetration using numerical IK.

2.5 Adaptation Versus Blending

Even though efficient techniques were developed to adapt a given motion clip to specific requirements, this approach still suffers from one major drawback: it can accommodate a given motion only to a certain point, otherwise the motion would get too much degraded. Indeed, if the desired motion is too far away from the clip that is adapted, then the corrections overwhelm the original motion, and the loss of quality becomes too noticeable.
Instead, the researchers tried to exploit the large amount of motion data that was captured in the recent years, like the Carnegie Mellon University database for instance [dat08]. The idea behind all these works is to reuse existing motion clips in order to create a new clip that would match the user requirements (figure 2.18). The problem thus becomes two fold. First the motions must be stored in such a way that they can be combined (or blended) together. Second, the motion data must be classified in such a way that the search in large databases remains possible, and computationally tractable.

2.5.1 Statistics and Graphs

Brand and Hertzmann [BH00] were the first to modify the way animations are usually stored. Instead of the classical parameter based representation of the human motions, they chose to use Hidden Markov Models (HMM). They extended the classical HMM by adding a Style variable, used for controlling the style into which a motion should be performed. As for regular HMM, the model learns itself from a dataset of examples, and is entirely unsupervised.
HMM are a probabilistic class of models; hence a sum of entropies (ambiguity in a probability distribution) and cross-entropies (divergence between distributions) is to be minimized, which would mean that the simplest model explaining the examples is found. In order to be able to generate various styles, the authors used two kinds of models. The first one aims at modeling the human motion in general. The other one is tailored to a given example motion. Thus, three quantities are to be minimized. 1. The generic model must be as closed as possible from the dataset. 2. The specific model must be as closed as possible from the generic one. 3. The models must be as simple as possible. The optimization phase is done via Expectation-Maximization (EM) which is a fast and powerful fix point algorithm for this kind of application.
The input quantities are the joints angles of the character alone, as they define the essence of a motion, which are pre-processed by a PCA analysis in order to remove their dimensionality and noise. Each style is thus encoded with HMM related quantities - namely state means, square root covariance and state dwell times. A new style can then be created by interpolation and extrapolation within this space.

Even though the use of HMMs to model motion has proven to be efficient, it remains a complex modeling process which cannot be implemented easily. Other ways to blend motion clips were thus investigated, and among them the idea of structuring a database according to the possible transitions between clips has been in the air for a while. What seems to be the right approach was presented in 2002 by several authors who published similar methods during the same SIGGRAPH conference.

Lee et al. [LCR+02] exposed a motion graph algorithm which uses a Markof process to implement the graph itself. First of all, a transition probability between each pair of frames from the motion database is calculated, based on a metric taking into account the joints orientation and velocity of the skeleton (improved later on by Wang and Bodenheimer [WB03]). The root translation and orientation is either taken as an absolute value or as a translation and rotation with respect to the previous frame, depending on the configuration of the virtual environment. For instance, if there are physical obstacles to be avoided, then the global frame is used. Second, the transitions are pruned according to various criteria to make the graph neatly connected (i.e. not too many transitions, no dead ends. . .)
A second abstraction layer was also added to make the search easier for a possible human interaction. Similar motions are grouped so that not too many potential transitions are presented when choosing for what the avatar will do next. Practically, the system can be used either using this second abstraction layer, or simply by drawing in 2D the path to be followed by a character. In this configuration, a search is performed which finds the shortest path along the graph making the character follow the user-provided path. Eventually, as the actual blending between motion clips during a transition is

likely to introduce artifacts such as foot skating, a retargeting stage is performed using Lee and Shin [LS99] method.

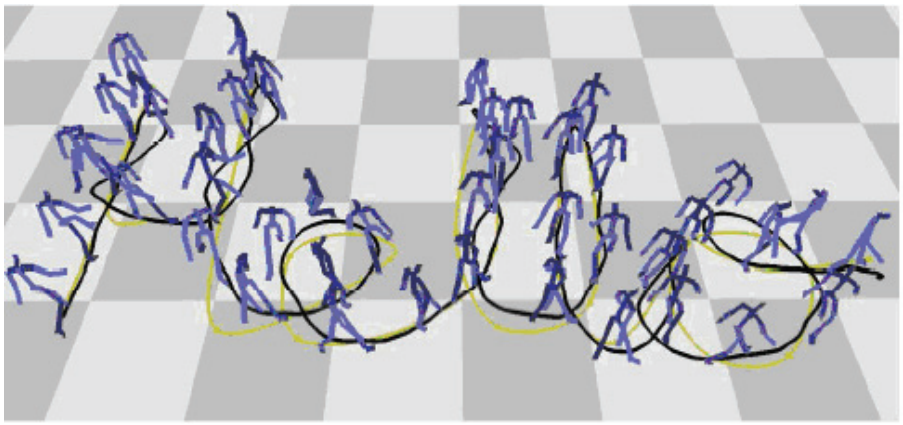

Figure 2.15: An example of an animation following a predefined path. The animation was created by Kovar et al. [KGP02]. Image from [KGP02] used by permission.

Kovar et al. [KGP02] approach was similar from the previous one from Lee et al. However, it features a few differences which probably makes it the most widely accepted implementation of motion graphs. The metric used for calculating the similarity between two motion frames does not take the velocities nor the joint angles as an input, but rather the position of points fixed to the skeleton over a time frame, which sounds equivalent. No choice is made between absolute or relative coordinates for the metric, but rather a closed form solution is used in order to replace the two motion sequences in the most similar configuration, hence discarding a possible global translation and rotation between the two clips. Transition selection is then simply defined by a threshold provided by the user. They prune the graph the same way Lee et al. does, however the motion database is augmented by adding reflections of the initial motions. The path synthesis is eventually done by finding a path along the graph which minimizes a function taking into account various requirements such as the path to be followed, or a specific clip to be used at a particular location (e.g. do a kick there).

The last work presented at SIGGRAPH 2002 which dealt with motion graphs is from Arikan and Forsyth [AF02]. The main originality of that paper compared to the previous ones lies in the path search algorithm. Instead of using global optimization, they adopted a trial and error approach. Paths are randomly crawled, and each generated path can be mutated in order to get a more optimized path which better matches constraints imposed by a user.
They further extended their work [AFO03] by adding annotations to the example database. Each motion clip is annotated by a user using an arbitrary vocabulary (walk, sit, turn. . .) this process being assisted by a Support Vector Machine (SVM). The user then specifies an animation timeline including the high level constraints such as walk or run, along with the previously available constraints of position and figure configuration for a given frame.

Li et al. [LWS02] also used a graph-like structure, this time coupled with a statis-

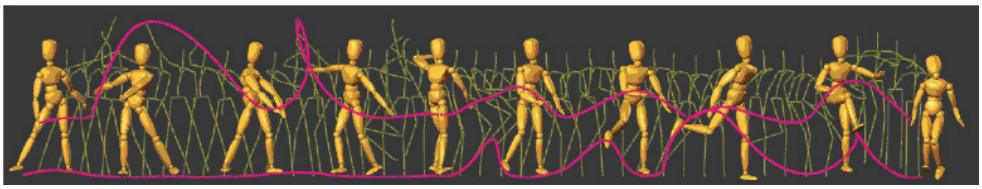

Figure 2.16: A dance motion generated by Li et al. [LWS02]. The system takes the start and end frames as a user input, along with the desired number of frames in between, and generates the entire sequence automatically. Image from [LWS02] used by permission.

tical model. Each motion sequence is cut into textons which encapsulate the dynamic properties of this particular sequence. Each texton has a hidden variable X of approximately twelve to fifteen components, along with an observation state Y which gives the real configuration of the figure. The motion is here modeled as a Linear Dynamic System (LDS) so that the specific dynamic feature of a particular motion can efficiently be identified and classified. Each texton is related to the others by transition matrices, thus creating the graph.
The resulting Markov model parameters derived by this approach are learned by finding a maximum likelihood solution. The only data provided by the user is the threshold of the data fitting error, hence controlling the length, accuracy and number of the textons forming the motion texture. Because LDS was used for representing the motions segments instead of the actual parameters of the original motions, the re-synthesized motion are truly originals whereas the previous approaches were mainly a concatenation of already existing motion sequences.

Gleicher et al. [GSKJ03] proposed an alternative way to construct simple motion graphs very much like the move trees well known in the video games community. The system inputs motion capture clips and under the guidance of a user constructs suitable connections and transitions between the animation frames. At runtime, the system displays the most appropriate frame according to the user input, and places the character at the correct location in space thanks to the stored displacement of the character for every frame of the database.

Eventually, Ikemoto and Forsyth [IF04] proposed to cut animations into pieces - one per limb - and then to assemble each motion segment differently in order to generate new motions. This approach was later improved by Heck et al. [HKG06] who took into account the lower body motion to adapt the transplanted limbs accordingly.

Very recently, Safonova and Hodgins [SH07] extended the motion graph approach by making possible to blend two motions synthesized by two different paths through the graph. This allows creating motions which did not exist before, with more variety than the classical graph approach. They also proposed a new method to prune the graph hence making it smaller, thus accelerating the searches.

2.5.2 Abstract Models

Kovar and Gleicher [KG03] used high dimensions curves - which they called registration curves - in order to encapsulate several parameters needed for the blending into one

single entity. Each registration curve thus gathers the time warp, spatial alignment and constraints between two motion clips. Time warping is optimized so that similar frames correspond to the same time once translated to generic time. In order to identify similar poses, the heuristic from Kovar et al.[KGP02] is used once again. Given two motions to be synchronized, a dedicated dynamic programming algorithm finds an acceptable path which crosses the frame difference map continuously and with acceptable slope. The frame difference map is a 2D map which for any of its point (i, j) gives the difference value of the frame i of the first motion and frame j of the second motion.
Frame alignment is performed by calculating the best alignment for each frame, and then fit a spline curve on the obtained values to make their blending easier. Constraints (e.g. foot planting) are kept only if the two motions being considered have the same constraint at the same generic time value. As the time step is small and the registration curves are smooth, the advancement of the root global location from a frame to the next one is done by estimating each derivative at that time for each blended clip, and blend these derivatives instead of the actual motion parameters. The actual integration is then performed with a simple Euler step. Eventually, if a constraint is to be enforced, it is done using standard retargeting techniques.
Heloir et al. [HCGM06] recently proposed an improved time warping approach, which they applied to sequences of the French Sign Language. Their new approach enables to obtain smooth time warping, without imposing constraints on the slope of the curve.

Egges et al. [EMM04] designed a framework able to complete the animated sequences of virtual human by idle motions generated automatically from - again - a motion database. Their approach exploits the blending techniques exposed in the previous works, but they did not construct a graph as idle motions tend to be similar from one to the other. Instead, the motion data is optimized through a PCA analysis so that the most important parameters are given more importance. Then it becomes easier to look at motion clips individually in order to check which ones are more likely to match for a desired transition between two motion clips. Each transition is chosen randomly among a set of predefined behavior (e.g. move the weight on the right foot, turn the head...), and the database of motion is checked in order to retrieve the best matched clip, i.e. the one which starts and ends by the most resembling postures.

Shin and Lee [SL06] reduced the dimensionality of motion through Multi-Dimensional Scaling (MDS), which has proven to be more suited than PCA for that purpose. The user can draw curves in the low dimensional space, which generates in return a motion by blending the poses associated with the similar curves surrounding the drawing. Each MDS is done on a small range of motion only, and several independent MDS can be put one after the other so that long curves can be drawn. This low dimensional space allows moving from one motion to another, thus constructing a kind of motion graph.

2.5.3 IK for Synthesis

While the synthesis of highly dynamic motions relied primarily on constrained optimization, manipulation tasks rather investigated the use of inverse kinematics for achieving its goals. As presented previously, many approaches made us of inverse kinematics for adapting a motion [LS99, TGB00, BLHB03] and there exists other approaches that use IK as the main animation engine. For instance, Yamane et al.

[YKH04] drive their IK model for animating a character performing manipulation tasks, and only use motion captured for optimizing a given posture so that it looks more realistic. More recently, Neff and Fiume [NF06] proposed a full hybrid IK system that utilizes optimization and analytic components. The purpose of such a system is to efficiently control all the parameters of the shape taken by a character, including his/her mood, expressivity and more. This approach was developed further [NF05] by including more animation aspects, and interaction language semantics for defining a clearer way of interfacing this system with the animator.

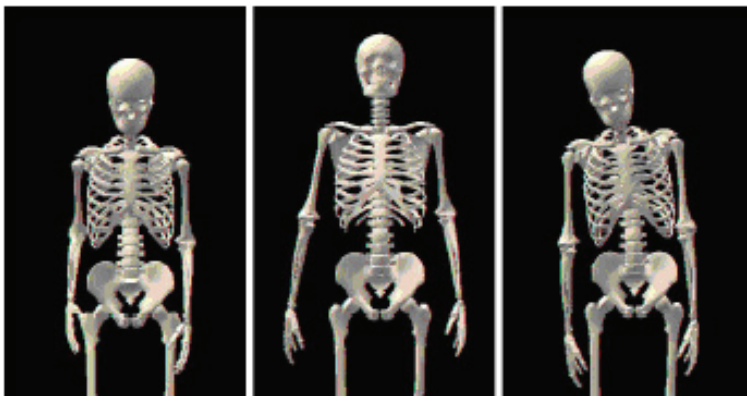

Figure 2.17: Three poses generated by Neff and Fiume [NF05]. From left to right: old man, energetic and dejected. Image from [NF05] used by permission.

2.6 Foot Skating Removal

Apart from the various approaches reviewed during the previous sections, a few extra ones were proposed to deal with the most common artifact to be removed when animating a virtual character: foot skating. This artifact stems from the way animations are dealt with in a computer. As explained before the character is placed in the 3D space using a global transformation applied to its root joint and its current configuration is obtained by applying a set of rotations to the joints. If the character being animated does not have the exact same proportions as the person who was motion captured, or if the captured data is noisy, then the movements of the legs do not exactly correspond to the translation and rotation of the root joint. Said another way, the global displacement of the character does not match any longer with the movements performed by its legs. The result of this is that one may witness the feet of the character sliding on the ground, or penetrating it (in case the animated character is taller than the original subject) or even floating in the air (in case the animated character is smaller than the original). This is an unpleasant and noticeable artifact; hence it has gathered the community attention which proposed several specific solutions for detecting when and where the foot must be planted, and how to efficiently modify the animation in order to comply with this constraint.

Of course, previously discussed method, mainly from sections 2.3.1 and 2.4.2 could also deal with this issue. However, because they are also able to deal with more complex behaviors, they tend to be complex and hence difficult to implement and to use. This is the reason why alternative, simpler approaches where developed in order to deal

with this specific issue.

Kovar et al. [KSG02] were the first to propose a dedicated solution to the foot skating problem. Their method uses a set of foot plant constraints that are either extracted automatically, or specified by the user. These constraints are then satisfied by smoothly changing the location of the root joint and the configuration of the legs. The root joint adaptation is done by calculating the locations that satisfy the constraints using reach spheres (i.e. the bounding sphere of the possible locations reachable by an effector). The limbs configuration adaptation is then performed using a variant of the inverse kinematics method proposed by Tolani et al.[TGB00]. Because sometimes the target can make the legs jerk in some circumstances, the limbs can be stretched by a factor of a few percents. This stretching is definitely the biggest drawback of their method, as most animation frameworks do not allow for such an operation.

Glardon et al. [GBT06] presented a method for efficiently detecting and correcting foot skating for motions generated on-the-fly. Their method requires that a few frames ahead of the current time are available to be able to start the posture adaptation before the foot planting constraint actually occurs.
For detecting foot plants, they rely on the height of the foot only (compared to other approaches which take into account speed as well) and dynamically set a threshold below which foot is considered planted. The threshold is estimated depending on the kind of motion being executed by the character (walk, run, sit...) by labeling manually the start and end of a foot plant, along with a value between zero and one in order to tweak the threshold.
The foot plant enforcement itself is achieved using the method presented in [BB04] with ease-in and ease-out phases during which the constraints are progressively applied. This stage uses cubic splines in order to blend the parameter curves, along with a more complex blending algorithm for handling cases where the motion changes drastically during the foot plant enforcement.

Ikemoto et al. [IAF06] proposed a framework which efficiently detects the foot planting of an animated character. They simply used a k-nearest neighbors classifier in order to label the data depending on the character's configuration (left toe planted, left heel planted, right toe planted, right heel planted). The system starts by asking the user to manually label a two hundred frames sequence. Second, it labels automatically - using the example data - another sequence of the database which is as different from the original sequence as possible. The user then examines the automatically labeled sequence and corrects the possible errors. The classification itself is then done by setting two thresholds on the confidence values of the foot planting in order to handle the cases when two values are close (when the weight is transferred from the heel to the toe for instance).

With the same goal in mind, Le Callennec and Boulic [LB06b] presented a robust approach for detecting three kinds of constraints for an articulated figure: space constraints (i.e. a point remaining stationary in space), line constraint (i.e. a point rotating around an axis) and point constraint (i.e. a point rotating around a point). For doing so, they express the transition between two consecutive frames as a displacement applied

on vertices, formulated as a local transformation. They then construct a linear system expressing whether or not a point remains stationary in space between two frames. They use two thresholds: σ_{max} is the maximal numerical value considered zero and ϵ_{max} is the maximal displacement considered null. The user gives a few template constraints which are used by the system in order to calibrate the thresholds using the least median of squares method [RL87]. One calibration per type of motion being processed is necessary, which is not too much of a constraint as the user simply has to specify the kind of constraint being active (e.g. left heel being planted) along with the corresponding time window.

2.7 The New Trends

Over the years, various directions were investigated while trying to make motion clips easy to use and to tune to specific user requirements. Some adapted existing clips, while others blended them together. Even though motion synthesis did not produce high quality results so far, this approach might be the future of motion adaptation. Indeed, it is clear that the best solution of all would be a system able to generate new, high quality motions which would match the user requirements. Even though such system does not exist yet, recent works investigated how it is possible to use existing motions in order to drive the synthesis of new clips (figure 2.18), thus ensuring both the compliance to user requirements and the high level of realism provided by motion captured clips.

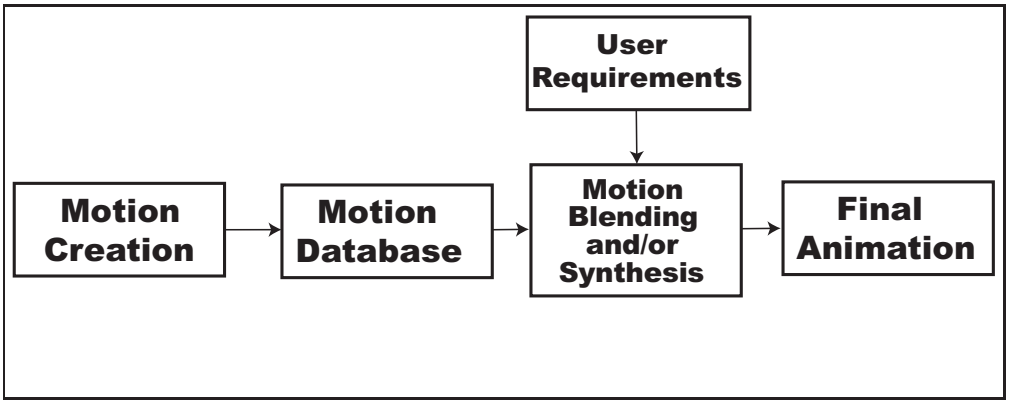

Figure 2.18: The newly emerging pipeline for motion authoring. First a database of motion is created regardless of the method employed to actually create the motions. Next the user requirements (e.g. follow this path, jump here...) are fed into a synthesis module, which often takes examples from the database in order to help the synthesis. Eventually, the final animation is created by blending existing clips from the database and/or by re-generating the motions from scratch.

Safonova et al. [SHP04] were among the first to exploit this paradigm. They improved the approach proposed by [FP03] to allow for the synthesis of motions: they reduced the search space by exploiting the correlation that exists among the human body while performing a specific action. The foreseen simulation is matched to a motion database which ensures that the result are visually pleasant.

Hsu et al. [HPP05] apply the style of an example motion to another motion clip, e.g. apply a sneaky crouch to a regular walking clip. Both the input motion and example target style must be similar, i.e. they must feature the same kind of movements like walking or jumping for instance. Both motions are put in correspondence by using a combination of approximate string matching techniques and time warping, which they call Iterative Motion Warping (IMW). This approach first adjusts the motion by adding or deleting frames of the motion so that they better match. Once a first correspondence is obtained, a global shift and scale are calculated to finely match both motions. The translation of style itself is done using a linear time-invariant model defining the relationship between the two motions. This mode uses four system matrices, used to perform a linear mapping between motions. These matrices are estimated using the N4SID algorithm [vd96]. Finally, artifacts might be created by the style translation, such as foot skating which they addressed using Kovar's approach [KSG02].

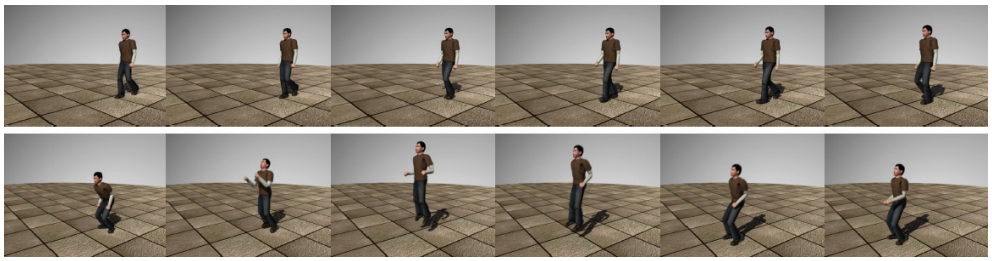

Figure 2.19: A walk motion created by Chai and Hodgins [CH07]. Even though the motion was generated from scratch, the natural look of it is due to the use of a motion capture database to *bind* the dynamic simulation. Image from [CH07] used by permission.

Very recently, Chai and Hodgins [CH07] constrained the dynamic system used to generate a body motion using a statistical analysis of an example database. This way, they were able to constrain the dynamic system within the properties of the example motion, thus ensuring that the generated clips are natural looking. The motion itself is generated by defining key poses over the animation, which are then matched through SQP optimization.
Sok et Al [SKL07] adopted the same philosophy by calculating the dynamic simulation counterpart of an example motion. They first estimate the simulations corresponding to various motions, and then they construct a finite state machine controller to drive the character. Using proportional derivatives control they are able to take the current state of the character, and to infer its next state by interpolating the k-nearest neighbors in the example data. Eventually, Da Silva et al [DAP08] proposed an approach similar to the one from [SKL07]. They also calculate the dynamic simulation corresponding to an example motion, as well as PD control to correct the results of the subsequent simulation so that it converges towards the desired state. However, they are able to alter a given animation clip by introducing new external forces, such as a character getting bumped into for instance.

2.8 Objectives

In the previous sections, we have seen that various approaches were proposed already to address several aspects of motion adaptation. High quality animations were produced even though user interaction is still needed to obtain the desired effects.

All these approaches considered the character's *dimensions* to perform the adaptation. However, to our knowledge, no work addressed the problem of adapting an animation according to the character's *shape*. Indeed, changing the girth of the limbs creates a special kind of self-penetration, which we will call *shape penetrations*. These penetrations are difficult to deal with because they arise between segments which are directly connected, and thus only one joint can be rotated to remove the penetration. Previous approaches relied on kinematic chains with several joints to address the *movement penetrations* and thus are not applicable here.

Our primary goal is to change the motion clip as little as possible so that no more shape penetrations remain, without introducing artifacts like foot skating or implausible physics. The methods we proposed here are meant to be applied *before* other classical adaptation methods, as depicted on figure 2.20. This way, it will be possible to animate unusual body shapes, without having a capture subject that somewhat resembles the target body. This used to be possible only to some extent, as the other adaptation algorithms addressed the movements penetrations and left to the designer the task of adapting the motion to the body shape.

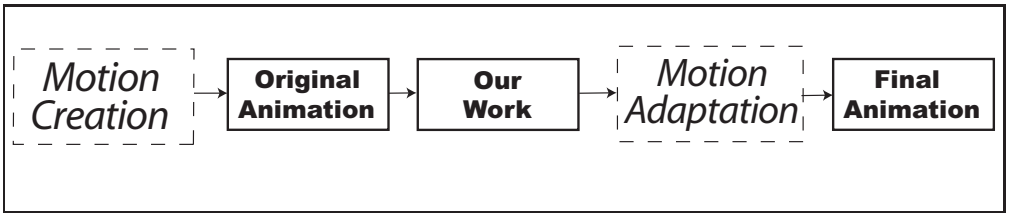

Figure 2.20: The same motion creation pipeline as on figure 2.3 augmented with our proposed approach. Our goal is to make any motion applicable to any character without the appearance of artifacts. Once an acceptable motion is obtained, the classical approaches of motion adaptation can be applied.

The approaches we expose in the coming chapters are not meant to modify the style of a movement as this was already addressed in [HPP05]. We rather aim at keeping the original style of the animation, and modify it so that a different body can be animated. Hence, when animating a T-Rex with movements coming from a ballerina, the T-Rex will still move like the ballerina, the only changes being meant to remove the self penetrations, footskating and correct the balance of the motion. This limits the scope of application of our method in the sense that if the target body is not capable of performing the desired motion in the real world, artifacts are likely to appear to cope with the unfeasibility of the motion.

Table 2.1 exhibits the specific features of the foot skating removal algorithms that were proposed so far. On this table, one can see that both [GBT06] and [KSG02] change the motion of the limbs during the process. Also, features such as resizing the skeleton

are totally forbidden because we want the dimensions of the character to be of a given size, making [KSG02] not usable for our goal. We tied out several approaches, such as analytic kinematics to perform the foot skate cleanup and select the one that better match our needs.

The main contribution we intend to bring is the ability to remove the foot skating artifacts from a motion clip *without* altering the movements of the limbs at all.

	Skeleton Resizing	Motion Altered	Path Preserved	Comments
Glardon06	No	Yes	Yes	Prioritized IK
Kovar02	Yes	Yes	Yes	Global optimization
Our Objective	No	No	Indifferent	

Table 2.1: Comparison of the features of the main foot skate removal approaches.

Another aspect of the work presented here is to adapt the motion of the character so that it better matches the shape onto which it is applied. This goal can be divided into two separate tasks: first make the character self collision free and second re-establish its balance so that the motion complies with the physical laws of motion.

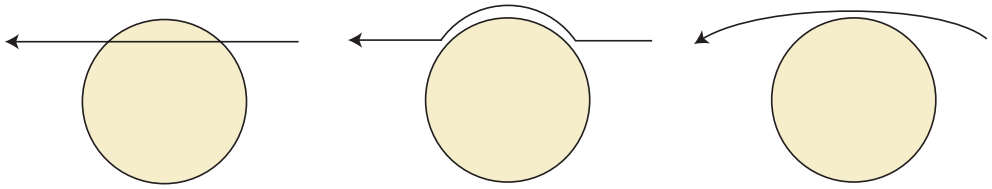

Figure 2.21: Illustration of the issue related to the per-frame approaches for collision removal. On the left is the original trajectory: The elbow joint is moving from the rear of the body towards the front, the circle representing a collision volume. On the middle a per-frame approach would apply a correction only when needed, thus discarding the natural movement. On the right is the effect we intend to produce, i.e. the motion was changed globally so that no more penetration remains, while keeping the resulting movement close to the original.

Several approaches where proposed already to address self-collision detection and removal [ZB94, PBCM05, JNK+02, JL00] as seen on table 2.2. However, all these approaches address collisions that are stemmed by the character movements. Such collisions happen during a short time interval only and mainly involve the end effectors; unlike collisions created by the character *shape* which tend to last for the whole animation range if not corrected. Even though [ZB94] and [JL00] approaches have proven

to be efficient for collision removal, they simply are not suited for this second kind of artifact.

The reason for this is the following: They modify the character pose on a per-frame

Author	Effect	Comments
Zhao	End Effector Penetration Removal	Potential Field Minimization
Peinado	End Effector Penetration Removal	Damping Zone around the volumes
Jeong	End Effector Penetration Removal	
Our Objective	Character Shape Penetration Removal	Our main goal

Table 2.2: Comparison of the features of various approaches dealing with penetration removal.

basis, thus placing the end effector right at the boundary of the collision volume. In the case of an arm swinging around a torso, this would make the arm strictly follow the torso's colliding volume and thus create an unrealistic motion (figure 2.21). We intend to present a way to deal with these *shape based* collisions so that they are removed, while preserving the motion characteristics.

Table 2.3 summarizes the main approaches dealing with balance enforcement. As we work on deformable characters, the model cannot be simplified otherwise many shapes would yield to the same adaptation, making Popović [Pop00] approach unusable. Several other works used a per-frame approach, which makes them cumbersome to use for dynamic motions. Shin et al. [SKG03] corrected the ZMP of the character with a per-frame approach; however we believe that this only works with well chosen motions. Eventually, Tak et al. [TK05] Kalmann filtering sounds compelling, but it requires that the filter is tuned for each kind of motion being adapted.
Our goal is to develop an approach that can adapt *automatically* an animation clip with no user interaction, and make it *at least* as physically plausible as the original input motion. By *at least*, we mean that the ZMP of the final animation should not be further from the supporting area than for the original clip.

In summary, the three main contributions that we intend to bring in this thesis are the following:

- A foot skate removal algorithm that do *not* alter the motion of the character's limbs nor the length of each segment.
- A self-penetration removal method that can cope with *shape* collisions while leaving the motion of the limbs close from the original.

Author	Effect	Comments
Boulic	Balance enforcement	Only the center of masses is brought back towards the supporting area
Popović	Balance enforcement	The character is simplified and this approach is far from running in real time
Shin	Balance enforcement	Per frame approach
Tak	Balance enforcement	A Kalmann filter must be tuned for each kind of motion, and the correction is done per frame
Our Objective	Balance enforcement	Automatic and global

Table 2.3: Comparison of the features of various approaches dealing with motion adaptation.

- A balance correction algorithm that can be applied to any body and motion, with no user interaction involved.

CHAPTER 3

Foot Skating Removal

Foot skating is an artifact commonly encountered when animating virtual characters. As explained in section 1.1 it manifests itself by the feet of the character sliding on the ground, if not flying in the air or walking inside the ground. Such artifact is noticeable by the casual eye, and thus this is the first aspect of the issue that we intend to address. Unlike previous approaches, which modify the character's pose instead of its path, we propose here a method that leave the animation of the limbs untouched.
Two different approaches were tried out during this thesis. The first one relies on the skin only to perform the correction, while the second one exploits a simple observation: the foot skate artifact comes from the fact that the character's hierarchy starts from the hip joint rather than from the point where the character is standing. Instead of using the usual root of the hierarchy, we make the hierarchy start from either feet, depending on the configuration of the character.

3.1 A Foot Skating Removal Method for Simplified Characters

Creating animations of virtual characters is a difficult task. It requires that a designer carefully tunes the motion so that it fit to some requirements, which usually demand hours of manual work. Once an animation clip is created, it tends to be reused as often as possible to avoid the heavy duty of designing a motion clip from scratch.
Once loaded in a virtual environment, motion clips may yet be altered to comply with newly introduced constraints, or by blending algorithms. Several institutions have developed their own VR platforms [PPM+03, Too08] which allow to re-use previously developed components. These frameworks are optimized to ensure maximum performances at runtime, and most of the models assume numerous simplifications in order to allow for rich environments. For instance, VHD++ developed jointly by the University of Geneva and EPFL does not allow to resize a skeleton at runtime, which leaves the method proposed by [KSG02] unusable within this context. Moreover, developers rarely focus on side artifacts and usually prefer to concentrate on the final user experience, which prevents them from implementing complex methods for little benefits. The

method we propose here complies with the two previous statements in the sense that it can accommodate rigid skeletons and is easy to implement. Thus it can be added with only little time and effort to an existing application.

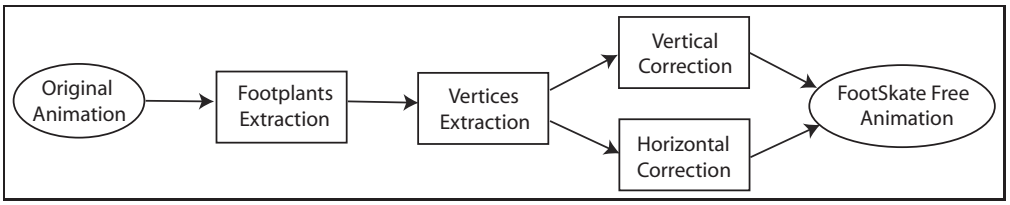

Figure 3.1: Functionnal drawing of the footskate removal method. First the planted feet and vertices are estimated, and then follows the vertical and horizontal correction, yielding to a new, skating free animation.

The method can be summarized as follow (figure 3.1): first foot plants are estimated, i.e. when should each foot be planted on the ground. Unlike most of the other approaches our algorithm does not constrain the *location* where a foot is planted, but rather the *frame* at which this should happen. This way, the motion itself remains as close as possible to the original, only the path followed by the character is subject to a scale.
The next stage is divided into two separate processes. A first treatment corrects the character's motion along the horizontal axis, and a second one adapts its vertical displacement. This choice was motivated by the observation that in many character animations, the feet remain rigid throughout the animations. This happens because the feet are attached to only one joint, again for optimization reasons. Thus, as the skin is not deformed accurately, the feet somewhat penetrate the ground regardless of the retargeting process applied. To correct this, our method accurately plant the feet where they should be in the horizontal plane, while in the vertical direction it minimizes the distance between the ground and the planted foot.

3.1.1 Feet Motion Analysis

Depending on the quality of a motion clip, it is hard to estimate how and when to plant a foot. If the motion is perfect, it should be enough to simply observe that a foot remaining static may be planted. However, feet are rarely motionless. Moreover most of the clips that are repeatedly used in reality are far from perfect and therefore such a simple criterion is insufficient. Previous works focused on proximity rules to extract the planting [BB98b], k-nearest neighbors classifiers [IAF06] or adaptive threshold imposed on the location and velocity of the feet [GBT06]. All the above mentioned approaches require some human interaction to perform the estimation: even [GBT06] requires to at least specify the kind of motion being performed. As we mentioned previously, our goal is to discard this interaction stage. Instead, we applied a two steps estimation taking the root translation and foot vertices into account. The first step finds out which foot should be planted while the second one refines which part of the sole should remain static. This is achieved by first extracting from the skin mesh the vertices for which the speed has to be calculated. We then isolate the vertices belonging to the feet by using the skin attachment data. Finally, we remove the ones for which the normal is

not pointing downward, which leaves us with the sole.

For clarity reasons, we use t to designate a frame index or a time interval, the unit corresponding to the actual time elapsed between two animation frames.

Foot Selection

Given the original root translation ΔR_t from time t to $t+1$ and v_i the vertex which is planted at frame t, we estimate which foot must remain planted at frame $t+1$ by considering the motion $\Delta R'_t$ for which the sole vertex v_j remains planted during the next animation frame. By planting a vertex at time t, we mean that its global coordinates remain constant during the time interval $[t-\frac{1}{2}, t+\frac{1}{2}]$. $\Delta R'_t$ can thus simply be expressed as:

$$\Delta R'_t = o(v_i, t) - o(v_i, t+\delta) + o(v_j, t+\delta) - o(v_j, t+1) \tag{3.1}$$

$o(v_i, t)$ being the offset at frame t of vertex v_i from the root, in world coordinates (figure 3.2). For this estimation, we take $\delta = \frac{1}{2}$ and $o(v_i, t+\delta)$ is calculated by linear interpolation between t and $t+1$.

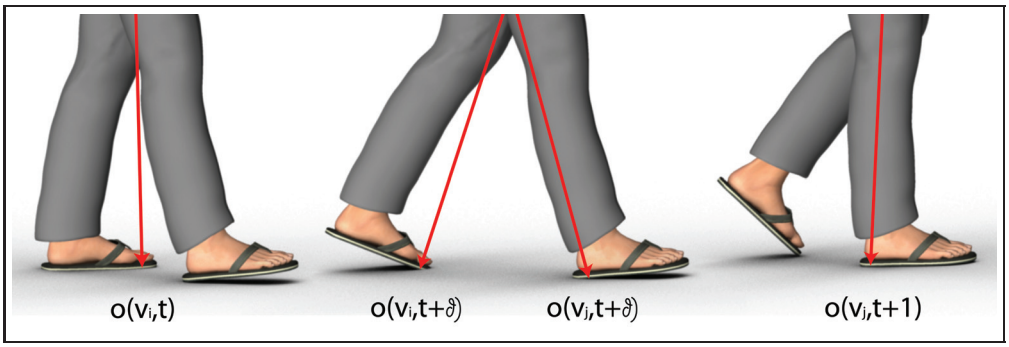

Figure 3.2: Offsets for foot skating removal. On the left, vertex v_i is planted, and remains so until $t+\delta$. At that time, vertex v_j becomes planted until $t+1$, i.e. the next frame. For clarity reason, more than one frame elapsed between the poses displayed on this figure.

Once we calculated $\Delta R'_t$ for all the sole vertices, we designate as static the vertex that maximizes the dot product p:

$$p = \frac{\Delta R_t}{\|\Delta R_t\|} \cdot \frac{\Delta R'_t}{\|\Delta R'_t\|}$$

Indeed, a higher value of this dot product means that if v_j is static at frame $t+1$, the displacement induced resembles more the original root motion. We discard the magnitude of the vector because we are interested in *where* the character is going and not *how far away* it goes.

Vertex Selection

The dot product criterion robustly tells us which foot must be planted. However, the actual vertex picked by this algorithm can sometimes be jerky, e.g. jump from the foot

tip to the heel. The reason is that we picked the vertex which keeps the motion as close as possible to the original one, possibly keeping a bit of skating on its way. In order to overcome this issue, we add a second selection process applied on the vertices of the planted foot only. This second process uses the *speed* of the vertices in order to pick the right one. Indeed, if the skating remains smaller than the root translation, then the vertex moving the less at a given frame should be made static. The speed of each vertex is first smoothed along several frames in order to remove some of the data noise (in our experiments, five frames appeared to be a good compromise). Second, the least moving vertex is chosen as the static one. The result of this selection over a foot step can be seen on figure 3.3.

We previously assumed that the static vertex in the previous frame must be known in order to estimate the one in the current frame. So for the first frame of the animation, we just use the speed criterion. One could think that the detection would be less accurate because of this, however we did not witness any setback during our experiments.

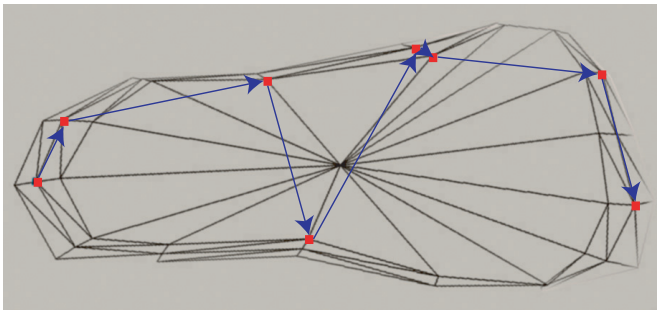

Figure 3.3: View of the trajectory of the least moving point over the sole during one foot step. In black is a wire frame view of the sole of the character, in red are the vertices selected during the step, and eventually the blue arrows shows the transitions between each point.

This algorithm has proven to be efficient on the animations we tried it on. It is even possible to discard the dot product phase of the algorithm but we noticed that this stage of the process significantly improved the robustness of the detection by accurately tagging which foot must be planted. In case our algorithm fails to figure out which vertex should be planted, one still has the possibility to manually label the vertices (or correct the output of the algorithm), as it is the case for all the previous motion retargeting methods. However, during our tests, this only happened on complex dance motions, for which it was hard even for the human eye to figure out which foot should be planted or not.

3.1.2 Root Translation Correction

As outlined in section 3.1, the adaptation is split into two phases, namely horizontal and vertical corrections. The horizontal correction introduces a drift of the character over the animation range in order to remove the foot skating, while the vertical processing aims at minimizing the distance of the static vertices from the floor. Using the dot product criterion for the vertices selection ensures that the drift induced by the horizontal correction changes only the *distance* travelled by the character and not *where* it is go-

ing. These two separate steps use completely different approaches, which are outlined in the next section.

Horizontal Correction

In order to calculate the corrected horizontal translation of the root joint between two frames, once again we use the motion of the vertices. In the previous section, we made the assumption that a static vertex remains so during a time interval of at least one frame, centered on the current time instant. However, due to the low sampling of the motion data which is often no more than 25Hz, this assumption cannot be retained for the actual displacement of the root joint. Thus, we estimate when the transition between two static vertices should happen, again using their speed. As we stated that the vertex with the less speed should remain static, we estimate the exact time instant between two frames when the transfer should occur.
For doing so, we approximate the speed of each vertex as follow: first the speed of the current and next static vertices v_i and v_j are calculated for frames $t-1$, t, $t+1$ and $t+2$. These velocities are then plotted in 2D and approximated using a Catmull-Rom spline [CR74] which yields to two parametric curves $V_i(q)$ and $V_j(q)$, $q \in [0,1]$, as depicted on figure 3.4. Eventually, the particular value q_t corresponding to the cross between v_i and v_j is calculated by solving the cubic equation $V_i(q) = V_j(q)$, which we did using the approach proposed by Nickalls [Nic93].

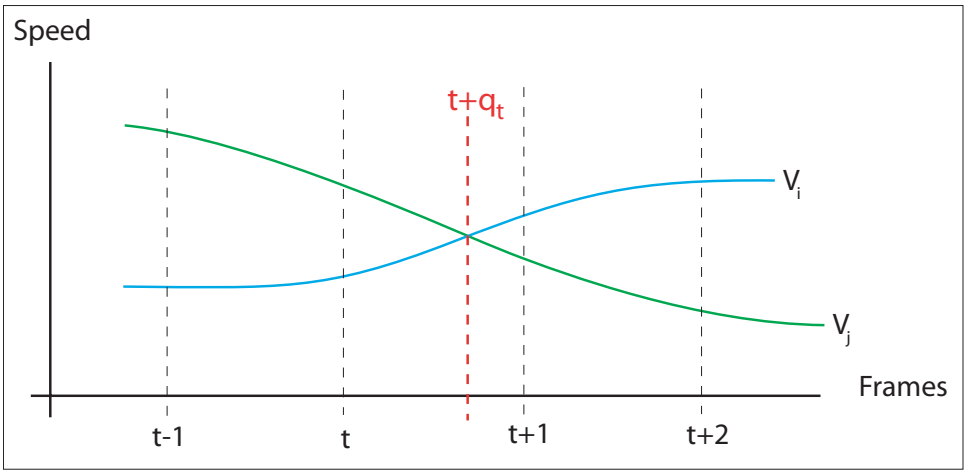

Figure 3.4: A conceptual view of the velocity estimation performed in order to determinate the exact instant of the weight transfer between two fixed points.

Now that the exact time $t + q_t$ when the weight transfer occurs is know, the actual position of the vertices at this instant is to be calculated (figure 3.5). For doing so, the trajectory of the points between t and $t + l$ is first approximated using again a Catmull-Rom spline. The parametric location t_1 and t_2 of the points over these curves is given by their approximated speeds as follow:

$$t_i = \frac{\int_t^{t+q_t} V_i(q)dq}{\int_t^{t+1} V_i(q)dq}, i = 1, 2$$

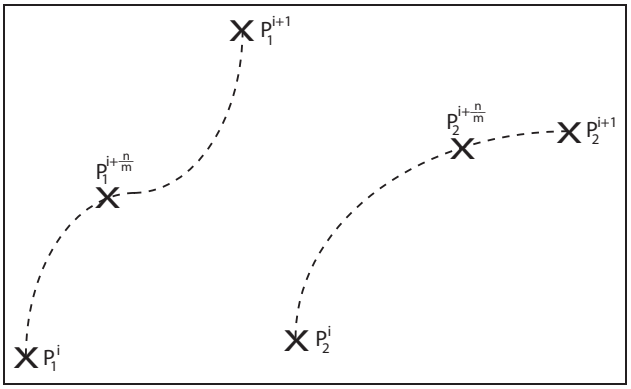

Figure 3.5: This figure illustrates the trajectory estimate that is performed between the points at frame i and $i+1$. m samples are calculated, and the n^{th} one is kept for the later calculation of the root translation.

Having the two offsets $o(v_i, t+q_t)$ and $o(v_j, t+q_t)$, enables us to calculate the new root displacement between frames t and $t+1$ using formula (3.1), with $\delta = q_t$.

The translation computed during this step is valid only if the feet deform in a realistic way which - to our experience - they seldom do. Often they remain rigid and this creates a bad vertical translation while the weight is transferred from the heel to the toe during a foot step. This is the reason why, as stated previously, the calculated root translation is only applied on the horizontal directions, as follow:

$$\Delta R_t^{horizontal} = P.\Delta R'_t$$

P being a 3D to 2D projection matrix.

3.1.3 Vertical Correction

The horizontal correction introduces some drift of the character compared to the original animation. This effect is desired as it removes the foot skating. However, in the vertical direction, no drift should take place otherwise the body will be walking in the floor or in the air. We do not want either to change the legs configuration for enforcing the correct height of the foot sole because we want to remain as close as possible from the original animation of the limbs. Moreover, strictly enforcing the height of the static vertices to be zero would lead to cumbersome configurations of the legs in order to cope with the rigidity of the feet thus introducing artifacts.
Instead we chose to act on the root joint translation only, by minimizing the height of the static vertices over the animation. Thus, a little bit of penetration remains afterwards, which is the price to pay if the feet are rigid and if we do not want to drastically change the look of the animation.
We calculate a single offset and a scale to be applied to the root height trajectory so that static vertices remain as close as possible from the ground throughout the animation, as shown on figure 3.

It is trivial to calculate the offset to be applied to the root trajectory: if we consider the height h_t in world coordinates of each static point, then the root offset ΔH is simply:

$$\Delta H = -\sum_{t=0}^{N-1} \frac{h_t}{N}$$

N being the number of frames of the animation.

Once this offset is applied to the root trajectory, the mean of the static vertices height is thus zero. However, they still oscillate above and underneath the ground during the animation. This oscillation is minimized by the calculation of the scaling factor α.

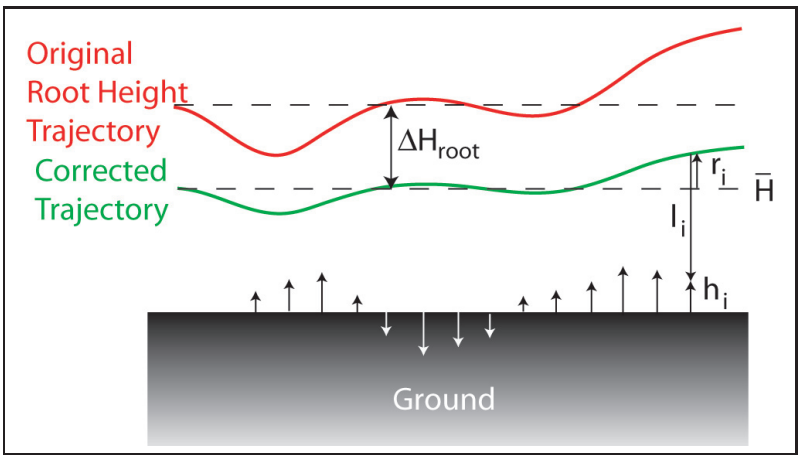

Figure 3.6: Scaling of the root joint height.

If we consider $\overline{H}$ to be the average root height over the animation, then for each frame its actual height H_t can be written as an offset r_t from this mean value: $H_t = \overline{H} + r_t$. The variance σ^2 of the static points heights can be expressed in terms of the root average height $\overline{H}$, the scaling factor α and the relative height l_t of the fixed vertex with respect to the root as follow:

$$N\sigma^2 = \sum_{t=0}^{N-1} h_t^2 = \sum_{t=0}^{N-1} (\overline{H} + \alpha r_t + l_t)^2$$

This variance is to be minimized by the scaling factor α, which this is equivalent to finding the root of a simple second order equation with only one unknown, α. Indeed:

$$N\sigma^2 = \alpha^2 \sum_{t=0}^{N-1} r_t^2 + 2\alpha \sum_{t=0}^{N-1} (r_t.(\overline{H} + l_t)) + \sum_{t=0}^{N-1} (\overline{H} + l_t)^2$$

As σ^2 and N are always positive, the minimal variance is given by:

$$\alpha = -\frac{\sum_{t=0}^{N-1} r_t.(\overline{H} + l_t)}{\sum_{t=0}^{N-1} r_t^2}$$

An alternative vertical correction algorithm was presented in [EBMT00]. They monitor the height of specific points of the foot sole (namely the toe and heel) and add a vertical translation in case a point falls below the floor, or if none of them is in contact with the floor. This correction is done per frame which might introduce artifacts in the

final animation. Our approach works globally hence it ensures that no high frequency signal is added to the motion. The drawback is that if the input motion is badly designed, our approach does not repair it and hence the feet might penetrate the ground a little bit.

3.2 An IK Based Approach

The method presented in the previous section gave good results. Indeed, as a long as the planted vertices extraction worked well, the skating cleanup has proven to be efficient. However, one might notice that the vertical correction does not completely get rid of the planted foot offset, but rather minimizes it. Thus, in some cases it may be possible that a foot does not stand on the floor, or penetrates it a little bit. To overcome this issue, we developed another approach to perform this cleanup, based on an adaptive hierarchy and analytic IK. This second method might modify the legs configuration, which is something we would like to avoid. However, the changes are small because the character's path is modified first.

3.2.1 Feet Motion

The philosophy of our approach is as follows: instead of being located always on the hip, the root of the skeleton animation should follow the actual walk and hence start either from the right or left foot depending which one is in contact with the ground (figure 3.7). This approach has proven to be successful when placing a character in awkward configurations, e.g. when hanging from its arms for instance. Badler et Al [BPW93] proposed to change the root of the hierarchy and to place it where the contact with the environment occurs, while Emering et Al. [EBMT00] extended this approach to make it compliant with multi-rooted skeletons. We propose to use it here for footskate cleanup.

Because most of the animation frameworks only support one node - the hip - as the root of the character animation, we compute a proper hip animation, along with the corrections that must be applied to the legs. This is done by going through all the animation once, modify the hierarchy on our way according to the feet motion analysis that was performed earlier, and re-compute the root transformation and legs correction.

For each frame, we check whether the foot in contact with the ground changed. If so, we then retrieve the current location of the new foot in contact with the ground and fix it. For the subsequent frames until the foot in contact changes again, we simply traverse the hierarchy from that point and retrieve the hip joint location and orientation which we take as the new correct one. The algorithm that performs this operation is as follow:

```
1: for all Animation frames i do
2:    fog ⇐ footOnTheGround(i)
3:    if footOnTheGroundChanged(i) then
4:       moveFootToTheGround(fog, i)
5:       rootTranslation ⇐ currentTranslation(fog, i)
6:       rootRotation ⇐ currentRotation(fog, i)
7:       hipRootMatrix(i) ⇐ getHipMatrixFromFoot(fog, i)
8:    else
```

```
9:        rootRotation ⇐ currentRotation(fog, i)
10:       hipRootMatrix(i) ⇐ getHipMatrixFromFoot(fog, i)
11:     end if
12: end for
```

This algorithm, even though trivial, has one line which is not obvious to perform: line four states *moveFootToTheGround(fog,i)* which means that the configuration of the skeleton must be adapted so that the foot onto which the body is standing on the next frame must be exactly placed at ground level. Thus, we developed a variant of the analytical formulation provided by Tolani [TGB00] in order to easily calculate the corrections.

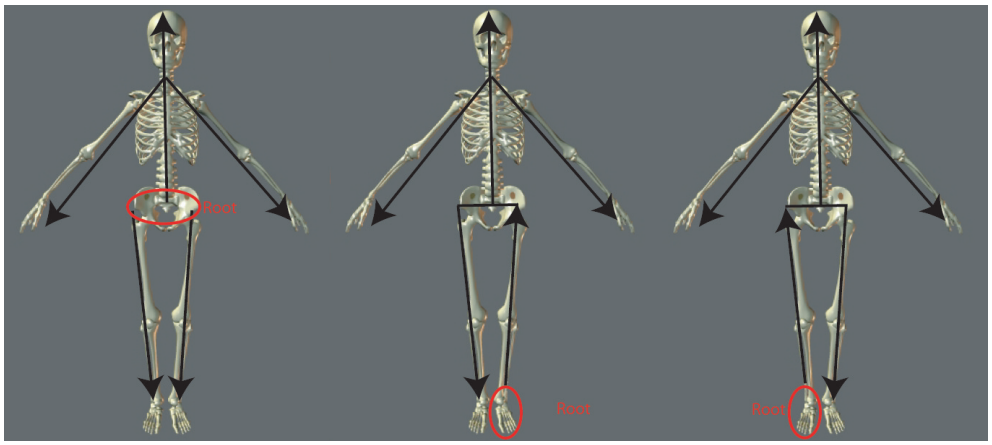

Figure 3.7: The three hierarchies that are applied on the skeleton, depending on which foot is in contact with the ground. The hierachy starts from the root node (circled red) and propagates through the skeleton following the direction of the arrows. Left is the regular hierarchy: the root node - usually located on the hip - is placed in the 3D space with a rotation and a translation, and the skeleton configuration is then adapted starting from that node. Center and right are the two alternative configurations, starting not from the hip but rather from the feet.

3.2.2 An Inverse Kinematics Method for Limbs Configuration Adaptation

Animators often have to adapt the configuration of the limbs - in this case the legs - of a virtual character in order to cope with the various bad configurations it may take. Inverse kinematics is very efficient to modify such a configuration, but it has one drawback: it must be carefully done by small steps otherwise the limbs are not driven towards their target configuration.

The special feature of the current problem that we exploit is the following: we would like to move the foot back on the ground while keeping its initial orientation. Moreover, while walking, the legs of the character stay almost in the plane defined by the hip, knee and ankle joints. Thus instead of working in the 3D space, the problem is reduced to the 2D plane to directly compute the corrections that must be applied to

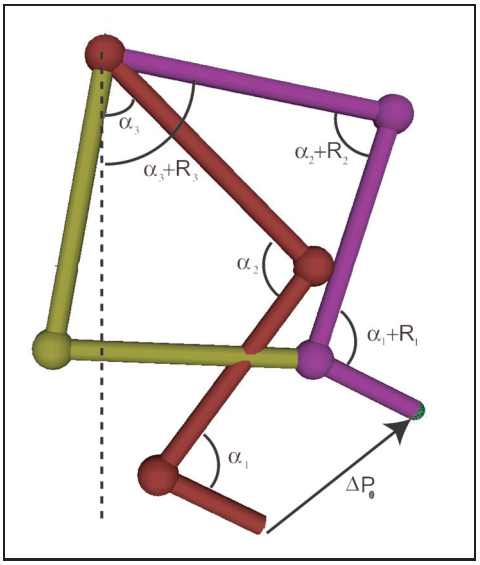

Figure 3.8: This figure illustrates what is done by our proposed method: in red is the default configuration of the leg and its associated angular values α_1, α_2 and α_3. We want to compute the angular corrections R_1, R_2 and R_3 that must be applied to these joints so that the end effector move by a vector ΔP_0 and that the last segment conserves its orientation. The yellow and purple segments are the two possible configurations that the leg can take in order to satisfy the displacement constraint.

these joints (figure 3.8).

In order to formulate the problem in a sound way, the vectors going from the centers of rotation to the end effector are considered (figure 3.9), as follow:

$$\begin{cases} V_1 = P_1P_0 = (0, y_1, z_1)^T \\ V_2 = P_2P_0 = (0, y_2, z_2)^T \\ V_3 = P_3P_0 = (0, y_3, z_3)^T \end{cases} \tag{3.2}$$

Developing from there enables to analytically compute the rotation angles which produce the desired displacement $\Delta P_0 = (0, d_x, d_y)^T$, as detailed in annex C. The values of α_1, α_2 and α_3 are thus given by:

$$\alpha_1 = sin^{-1}\left(\frac{y^2 + z^2 + R_1^2 - R_3^2}{2RR_1}\right) - k_1 - k \tag{3.3}$$

with:

$$\begin{cases} y = y_3 - y_1 + d_y \\ z = z_3 - z_1 + d_z \\ R_1 = \sqrt{(y_2 - y_1)^2 + (z_2 - z_1)^2} \\ R_3 = \sqrt{(z_3 - z_2)^2 + (y_3 - y_2)^2} \\ R = \sqrt{y^2 + z^2} \\ k = \tan^{-1}(\frac{y}{z}) \end{cases}$$

$$\alpha_3 = \sin^{-1}\left(\frac{y - R_1\cos(\alpha_1 + k_1)}{R_3}\right) - k_3 \tag{3.4}$$

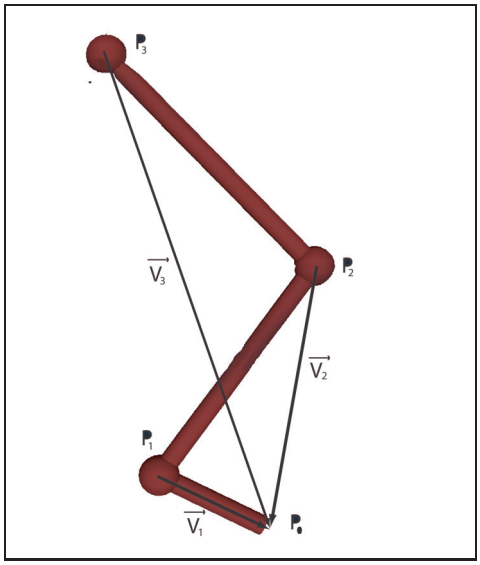

Figure 3.9: The three vectors going from each center of rotation to the end effector: V_1, V_2 and V_3 which are used for computing the final correction.

with:

$$\begin{cases} y &= y_3 - y_1 + d_y \\ k_1 &= \tan^{-1}(\frac{b_1}{a_1}) \\ b_1 &= z_2 - z_1 \\ a_1 &= y_2 - y_1 \\ a_3 &= z_3 - z_2 \\ b_3 &= y_3 - y_2 \\ k_3 &= \tan^{-1}(\frac{b_3}{a_3}) \end{cases}$$

Eventually:

$$\alpha_2 = -\alpha_1 - \alpha_3 \tag{3.5}$$

Practically, these values were used as approximated values in order to move the legs on the ground. First an intermediate frame defined by the hip, knee and ankle joints is computed, with its X axis orthogonal to the leg. Expressing the leg configuration with respect to this intermediate frame makes possible to compute the correction that must be applied to the joints.
Using the relative coordinates of this frame, a first computation of the corrections is done, as if it would completely lie in that intermediate 2D plane. These corrections are then applied to the joints, and the operation is repeated if one wants to get an even better configuration of the limb. Practically, most of the time the joints of the legs almost perfectly lie in a 2D plane hence only one iteration is enough for getting accurate corrections values. This approach is thus very fast: only two formulae give directly the correction.
This approach is similar to the one from Tolani [TGB00] but not the same. The main difference is that Tolani deals with a seven DoF limb in 3D, and goes through many

analytical steps in order to find the right solution. Our approach made simplifications (mainly the space reduction from 3D to a 2D) regarding the analytical part, which gives an accurate approximation very quickly. This estimate can then be refined via iterations, however we recommend to use Tolani's approach in case a high accuracy is required.

3.2.3 Solution Selection

For directly computing the values of α_1, α_2 and α_3 one has to use inverse trigonometric functions which provide two possible angle values for a given value of $\sin, \cos$ or $\tan$. Hence the output of the function must be disambiguated in order to retrieve the real value of the angle he/she is looking for.
The R-formulae proof (see annexe B for more details) tells us that $a = R\cos k$ and $b = R\sin k$. Moreover, $R = \sqrt{a^2 + b^2}$ hence R is always positive, thus $\cos k$ must have the same sign value as a and $\sin k$ the same sign value as b. k being defined as $k = \tan^{-l}(\frac{b}{a})$, the value one gets from $\tan^{-1}$ is always between $\frac{\pi}{2}$ and $\frac{\pi}{2}$. Fortunatelly, the `atan2` function available in the `C` standard math library returns the actual angle value depending on the signs of a and b, and thus we used this later function instead of the regular $\tan^{-1}$.
Besides, two inverse sinuses are also used for computing the values of α_1 and α_3 which yields to two values for α_1:

$$\begin{cases} \alpha_1 = \sin^{-1}\beta - k_1 - k \\ \text{or} \\ \alpha_1 = \pi - \sin^{-1}\beta - k_1 - k \end{cases}$$

and two more values for α_3:

$$\begin{cases} \alpha_3 = \sin^{-1}\gamma - k_3 \\ \text{or} \\ \alpha_3 = \pi - \sin^{-1}\gamma - k_3 \end{cases}$$

In these four solutions, the β and γ correspond to the values inside the parenthesis of equations C.11 and C.12. The first two values retrieved for α_1 correspond to the two possible configurations of the joints that drive the end effector hence they both can be considered as *plausible* and are disambiguate later on. However, from each of these solutions, the second inverse sinus again gives us two potential solutions and among these four values, only two must be kept (figure 3.10). This is done by re injecting the calculated values of α_3 into equation C.9: only two of the potential values satisfy this equation.

Now that only two solutions remain, one must choose which one is kept. This could be done by imposing boundaries on the rotation of each joint, but in our case, as the displacements that are applied are always small, the solution that has the smallest value for α_3 is retained.

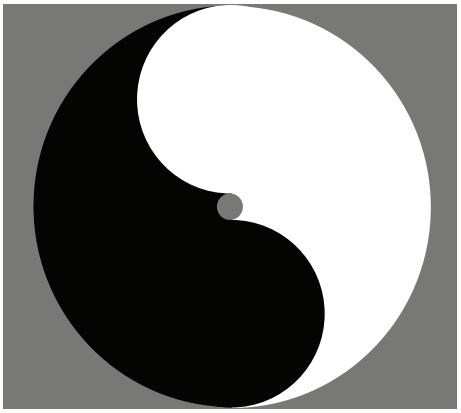

Figure 3.10: Yin-Yang like figure spawned by the solutions of our method. The plane was sampled, and depending of the validity of the first or second solution, the region was coloured black, white or grey. The black and white regions correspond to the area where a solution is valid and each color denotes the validity of one of the two solutions. The grey area denotes the region where it is impossible to reach the target.

CHAPTER 4

Character Based Adaptation

4.1 Introduction

There exists a wide variety of CG characters (figure 4.1) which are often animated with motion data captured from a real subject. Because it is unlikely that a T-Rex is available for a capture session, the motion data must be adapted to the specific character onto which it is applied. For instance, if the character has to grab a surrounding object, then the movement of its arm must be adapted so that he/she actually reaches the object.

Figure 4.1: A few examples of 3D characters. From left to right a human skeleton, an athletic man, a T-Rex, a plump lady and a skinny lady.

Deformable characters have been in the spotlight for a long time. They allow one to customize the shape of a body in order to reproduce the specific characteristics of a real person. Various approaches were proposed [SMT03, ACP03] and several companies from the clothing industry developed their own in-house solutions [Bro08, Opt08]. Until now, none of these commercially available packages were able to provide body animation as well as the deformations. The reason for this is twofold. First, the cloth simulations provided by these software packages are of high quality and thus would require a big amount of computation time. Second, a given body animation only matches the body for which it was created, therefore, when the body is deformed the animation should be adapted accordingly.

Many problems in the field of motion adaptation have already been resolved. However, most of the previous work focused either on the end effectors for interactions with the environment, or the physical properties of the motion that should be preserved. No method allows to adapt a given animation clip based on the actual shape of the character and this is what we intend to address.
Our adaptation process has two goals: to remove the self penetrations and to keep the physical properties of the motion. Our approach adapts the motion of each limb separately, and then corrects the balance of the character. This strategy has the advantage to divide the problem, thus making the adaptation easier to calculate.

We investigated two ways of handling the motion adaptation: spacetime optimization and Inverse Kinematics. We retained the first one because it allows handling the balance of the motion more easily than IK. Indeed, IK approaches act locally while spacetime optimization considers the entire motion to perform the adaptation.
The drawback of this feature is that spacetime algorithms are slow, preventing us from using it in real-time. To address this problem, we propose an interpolation scheme that allows to adapt the motion of a character in real-time from previously adapted examples.

4.2 Skeleton Design

To keep the adaptation framework as general as possible, we used the Biped hierarchy from character studio (figure 4.2) with thirty joints and twenty nine bones. The joints are not constrained, thus they can freely rotate around their axis and possibly go further than a real human could. This may be a problem when adding corrections on top of the existing rotations because there is no direct way to detect that a joint went further than what it can normally do. To address this issue, explicit angular values for each DoF must be calculated and clamped in case they reach an impossible configuration. In our implementation, we imposed the corrections to remain smaller than an arbitrary value defined empirically. Thus we do not enforce the joints limits explicitly, but rather forbid the adaptation to add big rotational values to the animation.

4.2.1 Limbs Simplification

The adaptation does not have to completely satisfy the penetration constraints. Indeed, the skin seldom deforms in a physical way but rather following interpolation schemes [MLT88, KCvO07]. Thus it does not make sense to strictly comply with the deformation, as an additional physical layer should then be added to account for the soft tissue deformations. Last but not least, the use of cylinders enables our algorithm to work even if the character deformation is of poor quality. Thus, instead of calculating the actual penetration distance between the limbs, an approximate value is computed and used in subsequent calculations. For doing so, we first match one cylinder per joint of the skeleton. Using a cylindrical model has two advantages: first it directly gives a fixed radius for each limb and thus eases the calculation of the self penetrations. Second, it provides an approximated volume and center of mass for each limb which are used for the balance correction.
Each average distance is estimated by first calculating the covariance matrix Σ_i for each

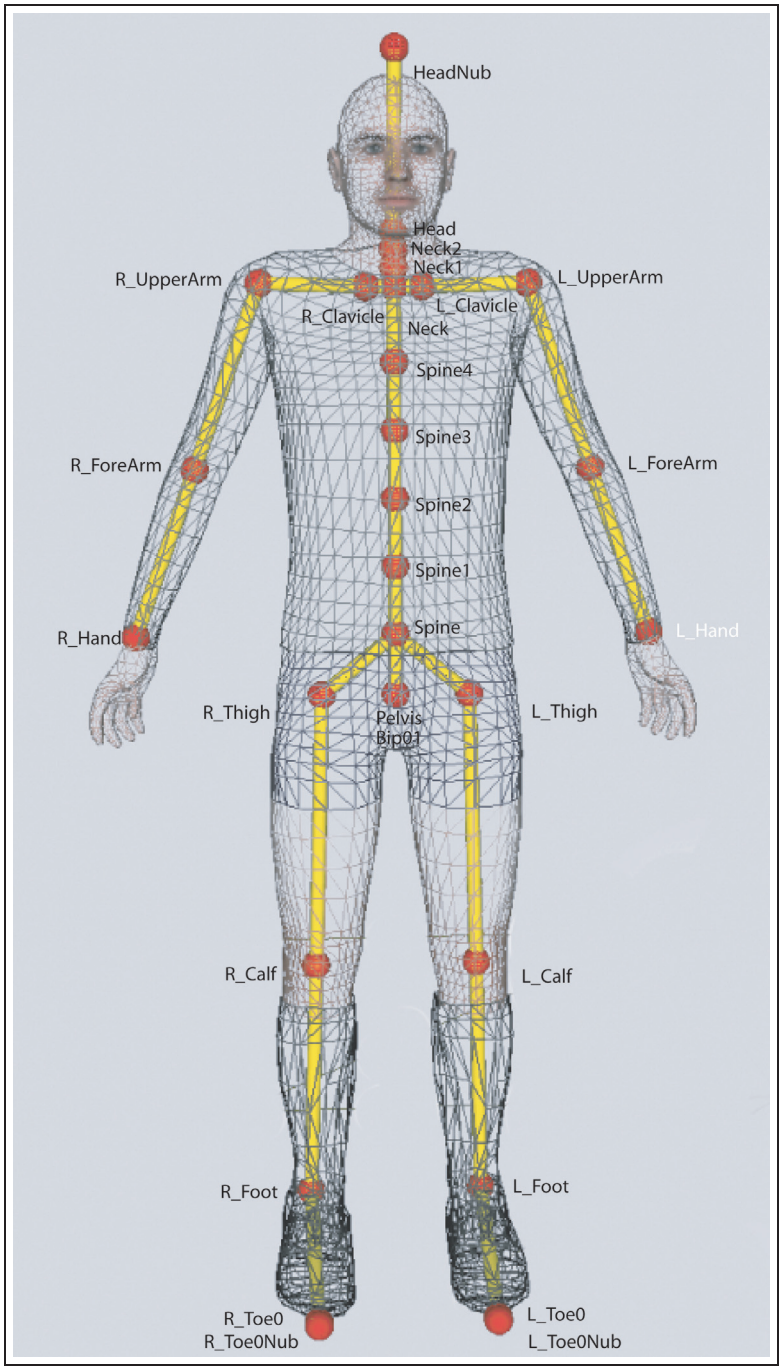

Figure 4.2: The biped hierarchy.

limb, as follow:

$$\Sigma_i = \sum_j \begin{pmatrix} x_j^2 & x_j y_j & x_j z_j \\ x_j y_j & y_j^2 & y_j z_j \\ x_j z_j & y_j z_j & z_j^2 \end{pmatrix} \quad j \in I_i^{bm} \tag{4.1}$$

Here x_j, y_j and z_j are the coordinates of vertex j. I_i^{bm} is the set of vertices to be considered for that particular cylinder translated so that its mean is zero.

Next, the eigenvectors and eigenvalues are calculated for each matrix. Most likely, the first eigenvalue and eigenvector correspond to the direction of the bone and the two remaining ones to the actual radius of the cylinder. This can be confirmed by discarding the eigenvector that gives the greatest dot product with the axis of the bone. Eventually, the radius of a cylinder can be taken as the average of the two remaining eigenvalues. An example of the generated cylinders can be seen on figures 4.3 and 4.4.

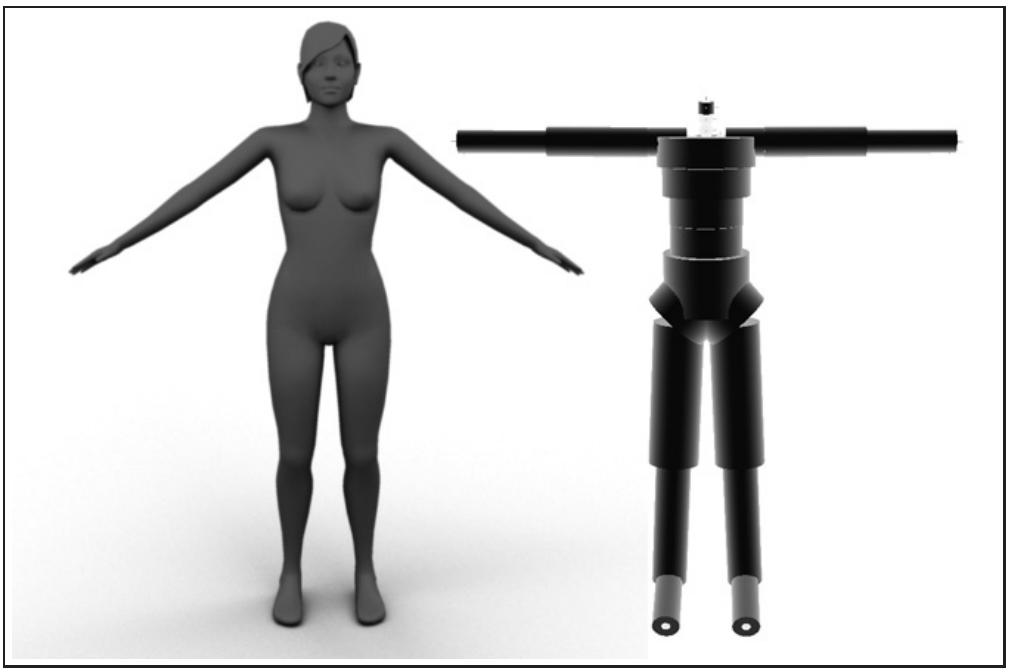

Figure 4.3: A virtual character (left) and its cylinders counterpart (right).

4.2.2 Vertex/Cylinder Allocation

The cylinders layout is predefined so that it neatly fit the body shape, as depicted on figures 4.3 and 4.4. The computation of the radius of each cylinder is done as follows: for taking into account every vertex of the skin, we must first find out which vertex will contribute to the radius of which cylinder. The criterion used for that is that if a vertex can be orthogonally projected onto the cylinder axis within its boundaries, then this vertex contributes to this cylinder. Because a vertex belonging to the arms project orthogonally on both the left and right arms cylinders, each vertex might get projected on more than one cylinder. Hence to figure out what is the correct cylinder to which a given vertex should contribute, the data from the skinning (i.e. how does the skin deforms with respect to the skeletal motion) is exploited for a first estimate, as follow :

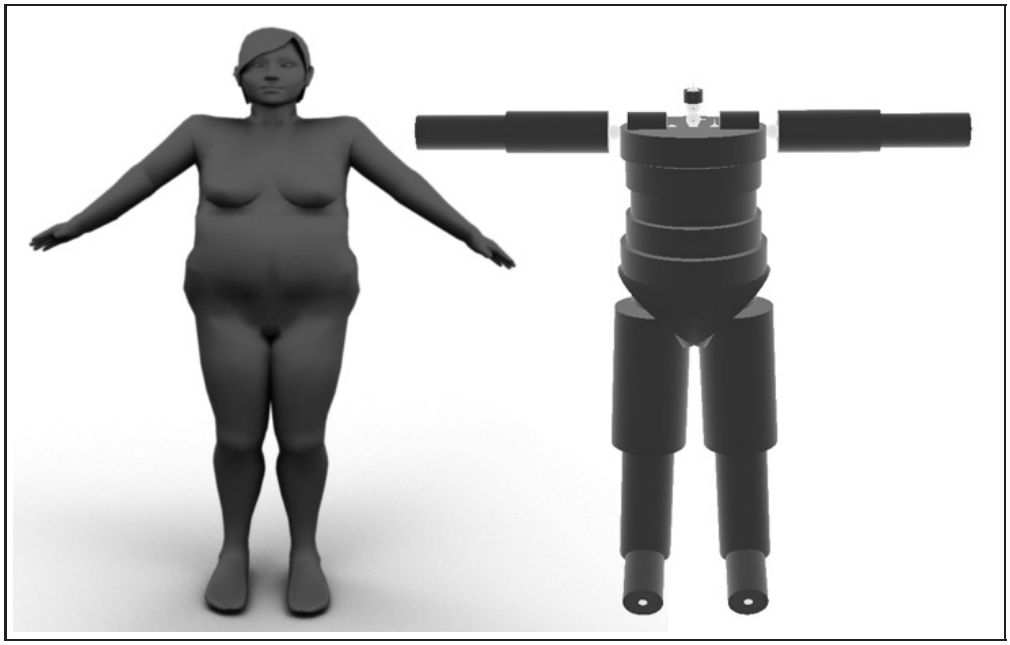

Figure 4.4: Another virtual character (left) and its cylindres counterpart (right).

$$c_i = Max(w_{i1}, ..., w_{in}) \tag{4.2}$$

With c_i the index of the cylinder to which the vertex number i might contribute, w_{ij} the weight of influence of bone number j to vertex number i and n the number of bones kept for the skinning deformation. A final check is then performed to find out whether the vertex projects onto its bone master or onto one of its two neighbors. Basically, the bone master of the skinning is kept, except if the vertex projects onto one of its neighbor, which is then selected instead of the master.

4.2.3 Penetration Calculation

The use of cylinders allowed us to speed up the process of detecting penetrations compared to a lower level approach such as collision detection on the skin mesh. However, it is not trivial to compute the penetration of a cylinder with a fixed length, and thus a fast and robust algorithm is given here. First, the minimal distance between the two lines supporting the cylinders is calculated. Each line is defined by a point and vector, as illustrated on figure 4.5. The minimal distance between these two lines can be retrieved by calculating two points: $P(s) = P_0 + s_c \overrightarrow{u}$ and $Q(t) = Q_0 + t_c \overrightarrow{v}$ which define the vector $\overrightarrow{w_c}$ perpendicular to both lines as follow:

$$\begin{array}{rcl} a & = & \overrightarrow{u}.\overrightarrow{u} \\ b & = & \overrightarrow{u}.\overrightarrow{v} \\ c & = & \overrightarrow{v}.\overrightarrow{v} \\ d & = & \overrightarrow{u}.\overrightarrow{w_0} \\ e & = & \overrightarrow{v}.\overrightarrow{w_0} \\ \\ s_c & = & \frac{be-cd}{ac-b^2} \\ t_c & = & \frac{ad-bd}{ac-b^2} \end{array} \tag{4.3}$$

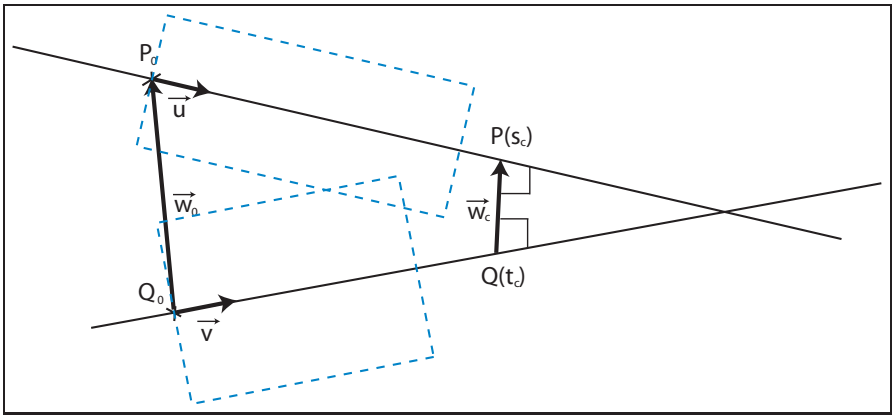

Figure 4.5: Computation of the minimal distance between two lines. Each line is defined by a pair point-vector (P_0, u) and (Q_0, v), and the minimal distance points $P(s)$ and $Q(t)$ are the points which define the unique vector w which is perpendicular to both lines. In dashed lines are the two cylinders supported by the lines.

Once the minimal distance points are obtained, the s_c and t_c are compared with the actual length of the cylinders in order to figure out which one should be kept as a cylinder, and which one should now be considered a disc. On figure 4.5, the bottom cylinder is considered a disc, and the top one kept as a cylinder. As soon as this configuration is obtained, the calculation of the closest point between the two cylinders is straightforward: the center of the disc C_d is projected onto the line supporting the cylinder, and these two points define a vector V_cl which is re-projected onto the plane defined by the disc to get another vector $V_discplane$. Normalizing this vector and multiply it by the radius of the disc gives us the closest point of the disc and from this closest point one can easily obtain the corresponding point of the cylinder.

Of course, this configuration is not always the one encountered when dealing with collision cylinders. However, reducing the problem to a cylinder/disc collision is always possible except when the closest point between the two lines supporting the cylinders actually lie on both cylinders.

As our skeleton is composed of thirty joints, tracking collisions would require that we calculate the penetration between all pairs of cylinders, which would dramatically slow down the calculations. Previous approaches [JNK+02] defined threshold distance between specific points of a body to calculate whether or not a penetration is taking place. As our cylinders are attached to the skeleton, we adopt the same strategy and only track the distance between joints of interest. This drastically reduces the computational burden of detecting collisions without sacrificing accuracy. The threshold distance between two joints typically is the sum of the two cylinders related to these joints.

4.3 A Global Optimization Approach

One of the most successful approach for modifying an existing animation is to use a global optimization algorithm. It has been used by various works [Gle98, PW99, SKG03] to address several aspects of the problem, such as foot plant enforcement, con-

straints compliance and physical properties of the motion. This approach uses a global optimization algorithm in order to modify the entire motion clip in one pass, unlike a per frame approach which would deal with each frame individually. The main advantage of the global approach compared to the local one is that it is possible to introduce a continuity criterion in the adaptation process, thus ensuring a smooth animation. Another benefit is that it is possible to address problems which require that several frames are taken into account to adapt a single frame.

One might be tempted to work directly with the recorded animation trajectories, and to modify them so that such or such criterion is satisfied. This has proven to be a bad idea for several reasons. First of all, these curves feature high frequencies coming from the actual character motion, which are of great importance for the natural look of the movement. These frequencies must be kept otherwise the adapted motion is degraded much. Thus working directly with these curves does introduce discontinuities in the optimization which would most certainly trap the optimizer in local minima. Second, depending on how the motion data is stored, it might take an extra step to extract the animation curves for each degree of freedom. If the skeleton rotations are stored using quaternions for instance, the rotation along each DoF must be estimated first to be usable in the optimizer. For all these reasons, it is best to add the an extra animation layer coming from the optimizer rather than dealing with the animation curves themselves (figure 4.6).

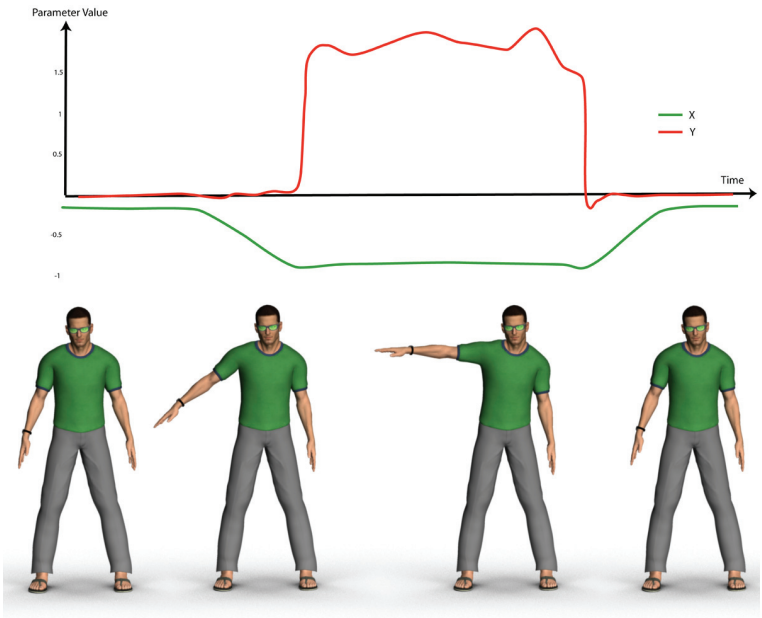

Figure 4.6: An example of animation curves. In green and red are the X and Y components of the quaternion rotation applied to the right upper arm of the avatar shown below.

The setup of the adaptation algorithm starts by defining the degrees of freedom (DoF) available to the optimizer. Each DoF allows the algorithm to modify the orientation of one joint along a given axis. A typical animation is several hundreds of frames

long, which would spawn hundreds of variables (one per frame) per DoF. A typical way to reduce the number of variables is to use control points of spline curves instead of their actual values. This approach also has the advantage to prevent the optimizer from adding high frequencies to the motion, which would make the final motion less realistic. The drawback is that the control points must be carefully chosen as the final result is dependent on them. In our experiments, we noticed that a way to place the control points is to roughly estimate where the minima and maxima of the corrections should be, and to place control points at these particular time instants.

The general problem of globally optimizing a motion can be formulated as follow:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} f(x) \\ &s.t.\ x^l \leq x \leq x^u \\ &\quad g_j(x) \leq 0,\ j \in I^n \\ &\quad h_k(x) = 0,\ k \in I^{ne} \end{aligned} \tag{4.4}$$

with x the vector of unknown, f the function to minimize [1] , x^l and x^u the lower and upper bounds of the variables, g_j the set of inequality constraints (would they be linear or not) and h_j the set of equality constraints.
In the motion adaptation process, x is formed by the set of DoFs available, f is the quantity to be minimized (e.g. difference from the original motion) while g and h are used to make sure that some constraints are kept during the adaptation process (e.g. keep the planted foot on the ground).

4.3.1 System Conditionning

According to equation 4.4 it is possible to assign a set of constraints over the variables. This works well if the starting point of the optimization lies within the feasible space for the constraints. In case the starting point does not comply with all the constraints, then the optimizer must generate a new starting point which can sometimes turn out to be difficult to do if not impossible.
We rather choose to use penalty functions in order to express the constraints. This works well for inequality constraints if the weights associated with each function are well defined. Equality constraints are more difficult to enforce due to the limitation they impose on the variables. This approach transforms equation 4.4 into:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} \alpha f(x) + \sum_j \beta_j p_j(x) + \sum_k \gamma_k h_k(x),\ j \in I^n,\ k \in I^{ne} \\ &p_j(x) = \begin{cases} g_j(x) & \text{if} \quad g_j(x) \geq 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{4.5}$$

The variables bounds can simply be enforced by clamping the variables to their $\min$ and $\max$ values. As we work with control points, clamping a variable to its limit does

[1] There could be more than one function to minimize, however this was left aside here for the sake of simplicity

not introduce a discontinuity. Indeed, the clamped value is smoothly interpolated between the surrounding control points, thus ensuring a smooth animation. At worse, the objective function does not be decreased enough to satisfy the constraints.

Even though well defined, large systems are hard to deal with. The high number of variables makes it difficult for the algorithm to figure out where to look for the optimal solution. Numerous valleys of local minima might trap the search, while peaks may prevent the algorithm from searching the entire solution space. There exist a way to break down a complex optimization problem into smaller, pieces called block coordinates descent [Coh92, Bet01, LHP06]. This approach assigns sets of variables that is fixed, while the other variables are being optimized. It works well if the variables are somewhat disjoint from each other, but it might have difficulties in finding the true minimum if the variables are too much correlated. The movements of a character is well separated from limb to limb, thus we adopted an approach similar to block coordinate descent. Each limb is adapted separately as shown on figure 4.7.

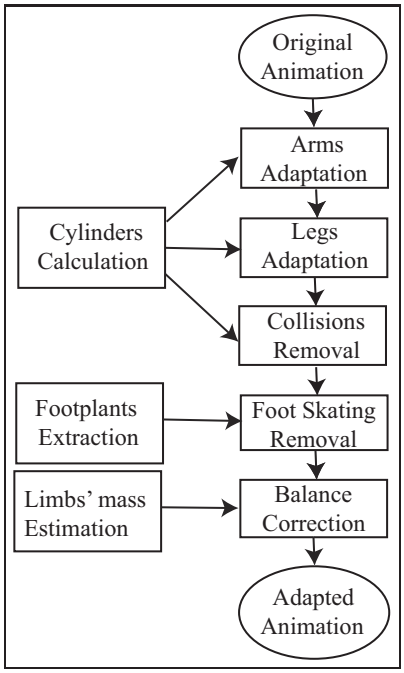

Figure 4.7: Conceptual representation of the motion adaptation.

Breaking the problem into smaller pieces does not only make the convergence faster, but it also drastically reduces the computation time. Indeed, because the character motion is pre-recorded, it is hard to find an analytic formulation for the derivatives of the motion. The derivatives of the motion are estimated through finite difference thus leading to a $\mathcal{O}(n^2)$ calculation at each step of the optimization, n being the number of free variables. Similarly to [Gle98], we do not act directly on the values of the parameter curves, but rather on the control points of splines. This extra layer ensures that the corrections do not add high frequencies to the motion, and it also reduces the number of dimensions of the parameter space. Regarding the spacing of the control points, for a walking animation it appeared that two control points per walk cycle are a minimum to efficiently adapt the animation. However, as it can be difficult to automatically extract

these cycles for any animation, one control point every two to five frames seemed to also work well.

4.4 Arms Adaptation

The adaptation of the arms motion is divided in two sets of variables: first the penetration itself is removed, and then the motion is adapted so that it better matches the original.

4.4.1 Penetration Removal

This stage of the adaptation simply consists in rotating the shoulder joints so that the elbow remains at a threshold distance from the torso. Because there is only one joint being rotated, there exist an analytical solution (c.f. 4.10.1) to this problem. However, analytical IK gives one particular solution per frame, which might introduce discontinuities in the resulting motion. Thus we choose to use a global optimization approach instead. The goal here is to keep the elbow joint enough far away from the trunk so that no penetration remains. As said in section 4.2.3 this is equivalent as keeping a minimal distance between two relevant joints, namely the *elbow* and *spine* in the case of a Character Studio compliant skeleton.

Objective Function

We know in advance that the input motion exhibits self penetrations between the arms and the torso as otherwise it would be pointless to trigger the algorithm. Thus the starting point of the adaptation does not comply with the basic constraint we would like to enforce. As said in section 4.3.1 it might turn out to be difficult - if not impossible - for the optimizer to find a good starting point. Moreover, we not only want to get a motion free of any self penetration, but we want also to change the joints angles as little as possible to keep the resulting motion as close as possible to the original. The objective function is the sum of two functions, one for the penetration removal and one for the minimal corrections, as follow:

$$\begin{aligned} f(x) =& \alpha x^2 + \beta \sum_m d_i(x) \quad x \in \mathbb{R}^n \\ d_i(x) =& \begin{cases} (||(P_i - Q_i)|| - d_{min})^2 & \text{if } ||(P_i - Q_i)|| < d_{min} \\ 0 \quad \text{otherwise} \end{cases} \end{aligned} \tag{4.6}$$

Here P_i and Q_i are the 3D position of the elbow and spine2 joints at frame i and d_{min} is the minimal acceptable distance between the two joints. α and β are meant to make the optimizer first remove the penetrations and then minimize the corrections. In order to achieve so, the magnitude of the function related to the penetration should be at least one order of magnitude higher than the minimal corrections function. In our implementation, the correction angles were expressed in radians, while the distances were centimeters, and thus both α and β could be set to one. x is a scalar representing the rotation of the shoulder joint. The rotation axis itself is dynamically chosen for each frame, so that the penetrations are removed with a minimal value of x. Considering a

penetration yielding to a desired displacement of the elbow joint $\vec{v}$ and the vector along the axis of the forearm limb $\vec{l}$ then the vector $\vec{r} = \vec{l} \times \vec{v}$ is the optimal axis of rotation for our task. Once $\vec{r}$ is calculated for every frame, it is kept constant for each frame throughout the optimization. Calculating one vector for each frame enables to drive the arm away from the body regardless of its current posture. Eventually, the use of the cross product ensures that the rotation axis is perpendicular to the upper arm's axis, thus the arm does not rotate around itself, leaving this DoF untouched for further adaptation.

4.4.2 Forearm Orientation Correction

The previous step made the motion of the arms self penetration free. However, its motion was changed and must be driven back towards its original configuration. What does being *close* to the original motion mean? It all depends of what one wants to achieve. For instance, it could mean that the end effectors remain at their original locations for grasping objects. As our focus was to keep the final animation as close to the original motion as possible, we choose to bring back the forearm towards its initial orientation (figure 4.8). Other objective functions might be devised if one wants to achieve other objectives, like grasp an object for instance. However, we believe that previous approaches already address this goal.

There are still several DoFs that remained untouched and it is over these ones that we will act. For the penetration removal two DoFs were used. Thus there are still one DoF of the shoulder and two more for the elbow that can be tweaked. At this stage, we do not use the DoF of the elbow which makes the forearm rotate around itself because it does not help to change its orientation. Here the objective function becomes:

$$f(x) = -\sum_{m}(H_i - E_i).(h_i - e_i) \quad x \in \mathbb{R}^n \tag{4.7}$$

H_i and E_i are the original hand and elbow locations at frame i, while h_i and e_i are the new locations after the adaptation and m is the number of frames taken into account. The ratio between α and β determines how far the forearm is brought back towards its initial orientation. Eventually, to make sure that the optimizer does not come up with big rotational values, constraints were added to all the available rotational DoFs to keep them between acceptable boundaries, as follow:

$$bg_j(x) = \begin{cases} -\frac{\pi}{2} - x_{\frac{j}{2}} & \text{if} \quad j \mod 2 = 0 \\ x_{\frac{j}{2}} - \frac{\pi}{2} & \text{otherwise} \end{cases} \tag{4.8}$$

The above set of constraints binds the variables between $-\frac{\pi}{2}$ and $\frac{\pi}{2}$. This does not apply for the elbow rotation, as it only can be rotated in one direction. For this particular joint, the constraints functions thus become:

$$g_j(x) = \begin{cases} x_{\frac{j}{2}} & \text{if} \quad j \mod 2 = 0 \\ -x_{\frac{j}{2}} - \frac{\pi}{2} & \text{otherwise} \end{cases} \tag{4.9}$$

It appeared that the objective function features peaks (figure 4.9) that prevent the optimizer from finding the true optimum in some cases. To address this issue, and

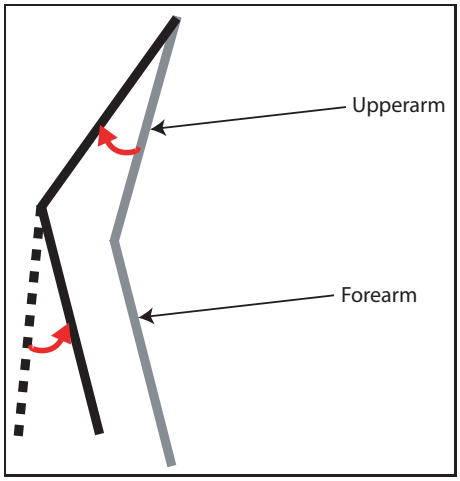

Figure 4.8: Illustration of the arm adaptation. In grey is the initial configuration of the arm. First the shoulder joint is rotated in order to drive the upper arm away from the body. The resulting configuration of the forearm has changed (dashed lines) and the elbow joint is thus rotated to bring the forearm towards its original orientation.

because only two variables are involved, we sample the search space every 0.1 radian and restart the optimizer from the ten smallest values of the objective function.

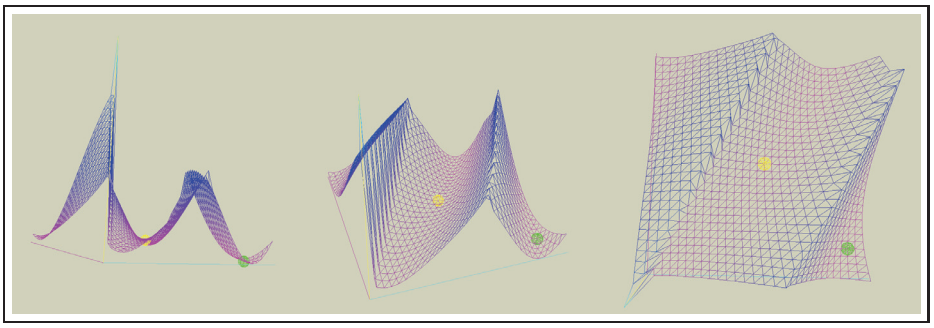

Figure 4.9: Plotting of the objective function from equation 4.7. From left to right: side view, perspective and top. The starting point with the parameter values set to zero is marked by the yellow sphere, while the true minimum is at the green sphere. In this case, the peak between the two locations prevented the optimizer from finding the true solution.

4.5 Legs Adaptation

In the same way the arms can self penetrate the body; the legs might have grown big enough to interpenetrate each other. Whether the penetration takes place on the thigh or calf does not greatly impact the algorithm we propose here. Again, for usual walking animations, detecting collisions between the two legs is equivalent to keeping a minimal distance between the calf joints. This is not true for more complex motions, e.g. if the character brings its foot towards its thigh, but we believe that in this case, a per frame IK solution might be more suitable.

Getting rid of the self penetration in the case of a walk does not allow for much freedom. Only the thigh joints can be rotated and the rotation of the hip can also help to move the legs apart (figure 4.10). Indeed, for a given posture and if the orientation of the legs is preserved, then rotating the hip helps to separate the legs from each other. Thus, three articulations (left thigh, right thigh and hip) are available for taking care of this penetration. Among these three only two are kept because opposite correction values should be applied to the legs.

The objective function associated with the legs resembles the one used for the arms penetration removal:

$$\begin{aligned} f(x,y) =& (\alpha x)^2 + (\beta y)^2 + \gamma \sum_m d_i(x,y) \quad x,y \in \mathbb{R}^n \\ d_i(x,y) =& \begin{cases} (||(L_i - R_i)|| - d_{min})^2 & \text{if } ||(L_i - R_i)|| < d_{min} \\ 0 \quad \text{otherwise} \end{cases} \end{aligned} \tag{4.10}$$

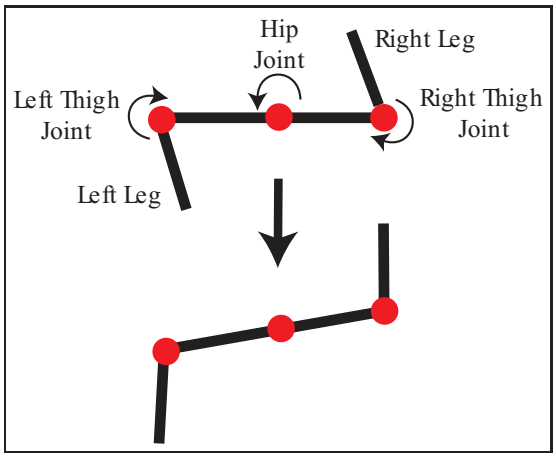

Figure 4.10: Conceptual view of the legs configuration adaptation. The modifications are applied to the hip, left thigh and right thigh joints in order to drive the legs away from the body.

This time, x represents the values of the hip rotation control points and y the thigh rotations, L_i and R_i are the left and right knee positions at frame i.
The squared distance used for the optimization works well, but it fails to efficiently discriminate cases where the self collision might appear in the upper legs region. This can be addressed by replacing the distance function of equation 4.10 by the following:

$$d_i(x,y) = \begin{cases} (||M(L_i - R_i)|| - d_{min})^2 & \text{if } ||M(L_i - R_i)|| < d_{min} \\ 0 \quad \text{otherwise} \end{cases} \tag{4.11}$$

Here M is a matrix that projects the considered points onto the plane defined by the vertical axis and the line between the left and right thigh joints. Conceptually, this has the effect of taking into account the actual separation of the legs, would they be close by (e.g. while standing) or rather far apart (e.g. during a walk).

A penetration might be removed by increasing the x or y values separately, thus the ratio between α and β determines whether it is the hip or thigh rotations that take care of the issue. In our experiments we noticed that a ratio of about ten (i.e. $\beta = 10\alpha$) produces results which look natural.

4.6 General Purpose Collisions Removal

The arms and legs adaptation removed the penetrations related to the character shape. The remaining penetrations occur during a shorter amount of time, when the character puts its hand on its waist for instance. It is thus possible to use existing IK approaches such as [BLHB03, JL00] to correct this. However, as we have the optimizer available to us, we propose to reuse it for this purpose. The principle remains the same, i.e. modify a given set of DoF so that no more penetration remains. The objective function becomes:

$$\begin{aligned} f(x) =& (\alpha x)^2 + \beta \sum_J p_{ij} r_j^2 - d_{ij}^2 \\ p_{ij} =& \begin{cases} r_j^2 - d_{ij}^2 & \text{if } r_j^2 - d_{ij}^2 > 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{4.12}$$

With x gathering the corrections of the DoF chosen for the adaptation, J the set of cylinders taken into account, r_j the radius of the j^{th} cylinder and d_{ij} the distance at frame i between the colliding joint and the line supporting cylinder j. Better than a joint, it is straightforward to define one or several vertices of the body mesh that should not penetrate the body. Each vertex becomes an offset from its master joint, thus preventing a full deformation of the skin to calculate the value of f.
We implemented this algorithm and took into account the legs/torso, hands/torso, hands/head, hands/shoulder and hands/legs penetrations. We allowed the character to bend its elbow, rotate its shoulder and bend its torso depending on the kind of penetration considered. We placed control points every two frames because these collisions can occur within a short time.

4.7 Balance Correction

After the arms and legs penetration removal, the motion of the character has slightly changed and its balance is not maintained any longer. One last step takes place, so that the movement of the character complies with the physical laws of motion.

The Zero Momentum Point (ZMP) is of great importance to assess the balance of a character. Its definition somewhat differs from publication to publication, but here is a way to think about it. When a character moves, the movement of its body creates momentum which yields to a force f and momentum m applied to the last body before the ground contact (usually the foot in contact with the ground). f and m must be compensated for, otherwise the character would collapse.
$f = (f_x, f_y, f_z)^T$ and $m = (m_x, m_y, m_z)^T$ can be split into their horizontal ((f_x, f_y) and (m_x, m_y) resp.) and vertical (f_z and m_z resp.) components. The horizontal components (f_x, f_y) of the force and vertical momentum m_z can be accounted for by friction,

as long as the foot remains in contact with the floor. The remaining components, however, must be compensated by the force generated by the ground in response to the foot pressure. This force can only be applied somewhere on the foot sole, and as the character has a given weight its magnitude is also limited. Thus, the point where this force must be exerted *has* to lie within the supporting polygon, otherwise the dynamic balance cannot be maintained. An extensive definition and discussion regarding the ZMP concept can be found in [VB04].

An active human body is rarely in equilibrium [Hud96], and thus it often happens that this point goes out of the supporting polygon. Such a configuration does not mean that the person will necessarily fall, but rather that he/she must move his/her feet somewhere else before he/she collapses to the ground. For instance, during a normal walk cycle the ZMP often goes out of the supporting area for a short instant. If something prevents the person from placing his/her other foot in front of him/her (step on the shoelace for instance), then he/she would actually fall straight to the floor. Oppositely, if the ZMP is inside the supporting area, this means that he/she has the ability to move towards an equilibrium state without having to reconfigure his/her foot plants. Because the momentum cannot be compensated for when it is outside of the supporting area, this point is often referred to as the Fictitious ZMP (FZMP).

In the remainder of this document, we call the point where the force should be applied to compensate for the momentum regardless if the ZMP lies within the supporting area or not. Robotic applications must maintain the ZMP within the supporting area otherwise their prototype would fall to the ground, and possibly get damaged. VR applications are different in the sense that even if the motion is not balanced, the character does not fall. For us, the ZMP can be seen as an indicator of the balance of the character.

The ZMP can be difficult to calculate, but there exist a closed form solution [TK05] for doing so:

$$ZMP = \begin{pmatrix} \frac{\sum_i m_i(\ddot{y}_i+g)x_i - \sum_i m_i\ddot{x}_i y_i}{\sum_i m_i(\ddot{y}_i+g)} \\ \frac{\sum_i m_i(\ddot{y}_i+g)z_i - \sum_i m_i\ddot{z}_i y_i}{\sum_i m_i(\ddot{y}_i+g)} \end{pmatrix}$$

with $\mathcal{R}$ the set of rigid bodies composing the character, and m_i being the mass associated with rigid body i. Due to the second derivative terms, this calculation involves several consecutive frames, thus a spacetime approach is well suited.

Before acting on the physical properties of the character, the weights of the individual limbs must be estimated, along with the duration of the balanced and unbalanced states. This preprocessing is explained in the two following section.

4.7.1 Weights Estimation

To perform an accurate adaptation of the gait of the character, the weights of each of the limbs must be known. In the case of a captured motion, the total weight of the subject being captured can be acquired at the same time, but even then this is not the data we are looking for. Moreover, assuming that the muscles of the character can exert an arbitrary force, then the balance of a character is only bound by the distribution of its mass over its body and not by its total mass.
Thus the individual mass of the limbs must be estimated. It is very difficult, if not

impossible, to accurately measure the mass directly on the subject being captured. Because this issue arises as soon as one tries to deal with the physical properties of a movement, several studies were conducted in the past to get a meaningful estimate of the density of the limbs. For instance, the US army conducted series of measurements on corpses [CCM+75, Dem55] and eventually came up with the density measures reproduced on table 4.1 and masses percentages.

Body Part	Density (in grammes per milliters)
Head	1.056
Torso	0.853
Upperarm	1.0045
Forearm	1.052
Hand	1.080
Thigh	1.0195
Calf	1.069
Foot	1.071

Table 4.1: Mean density of body parts taken from [CCM+75].

Using this table and assuming that a particular limb's volume is known, it becomes straightforward to calculate the limb's mass. Previous works used global optimization techniques in order to estimate the body mass [SKG03]. They calculate the optimal mass distribution so that the original motion clip would appear to be balanced throughout its duration. However, we noticed that this optimization easily fails to accurately estimate the mass, and often falls into local minima with an unrealistic mass distribution. Instead we decided to calculate this data from the volume of the limbs along with the densities from table 4.1.
Another way to proceed would have been to calculate the entire volume of the body using for instance the method proposed by Muller [MHHR07] and additionally to use the mass distribution table provided by [Dem55] to allocate a specific percentage of the total mass to each limb. Again, the actual total mass of the character is of little interest here because the balance is not modified by a global scale applied to the weights. For the calculation of the volume of the limbs, we relied on a cylindrical model of the body [HE64] because it also provides the center of mass associated with each limb. This enables the direct calculation of the ZMP of the character from a given animated clip.

4.7.2 Threshold Distances Calculation

During dynamically balanced motion, it is commonly admitted in the CG community that the ZMP should remain within the supporting area of the character. This, to our experience (c.f. section 5.6) is not always the case. Thus, instead of bringing the ZMP back within this polygon, we calculate it for the original motion and body shape. This gives one distance zmp_i between the ZMP and supporting area for each frame i, distance which is used as a per-frame threshold during the balance correction.

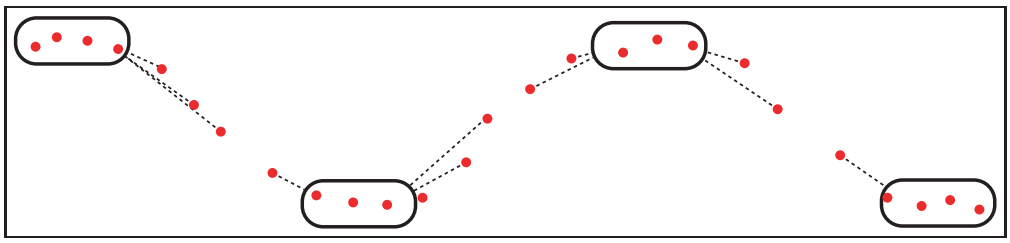

Figure 4.11: Illustration of the balanced frame concept. In black are four successive supporting areas, and in red are several calculated ZMPs. The dashed lines are the threshold distance zmp_i that are kept for each frame of the animation

4.7.3 Balance Optimization

The goal here is to bring back the ZMP closer than zmp_i from the supporting area (figure 4.13). For doing so we move the Center of Mass (CoM) of the character using the ankle and thigh articulations or lean the torso (figure 4.12). We allow the optimization to use both these strategies by defining four free sets of variables: two for the legs adaptation and two for the torso. Each pair of variables make the CoM move sideways

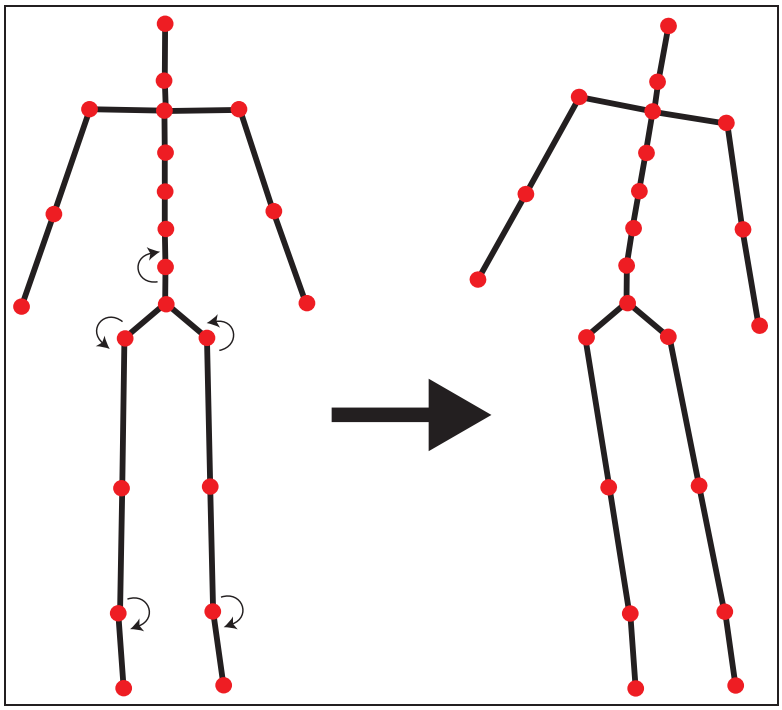

Figure 4.12: Conceptual view of the balance adaptation. The legs move the CoM towards the left, while the torso is bent so that the CoM moves in the opposite direction.

or forward/backward. The torso variables can directly be applied to the spine joint of the character, while the legs variables are shared by the ankle and thigh joints with the appropriate sign value. Because a distance from a supporting area is analog to a penetration, we can reuse the same optimization strategy. Here the objective function

for the balance becomes:

$$f(x) = \alpha||x||_1 + \beta \sum_m d_i(x) \quad x \in \mathbb{R}^n$$
$$d_i(x) = \begin{cases} ||(S_i - \text{ZMP}_i)|| - zmp_i & \text{if } ||(S_i - \text{ZMP}_i)|| > zmp_i \\ 0 \quad \text{otherwise} \end{cases}$$

ZMP_i is the location of the ZMP at frame i, S_i is the closest point from the ZMP on the supporting area and zmp_i is the maximal distance from the supporting area allowed for the ZMP.
As for the arms constraints, some local minimum of $f(x)$ might be achieved by non realistic (and impossible) configurations of the limbs. Hence we also impose bounds on the values of x, i.e. the corrections applied to the motion cannot exceed $\pm\frac{\pi}{8}$ as follow:

$$g_j(x) = \begin{cases} x_{\frac{j}{2}} - \frac{\pi}{8} & \text{if} \quad j \mod 2 = 0 \\ -x_{\frac{j}{2}} - \frac{\pi}{8} & \text{otherwise} \end{cases} \tag{4.13}$$

In case the original animation with an appropriate character shape is not available, the threshold distances zmp_i can be roughly estimated from the current animation and character shape. In this case, if the corrections are not sufficient from the user's point of view, applying a scaling factor less than one to the zmp_i increases the amount of correction. In our experiments we noticed that a factor below 0.9 produced cartoon like animations, with the character leaning much more than it would normally do.

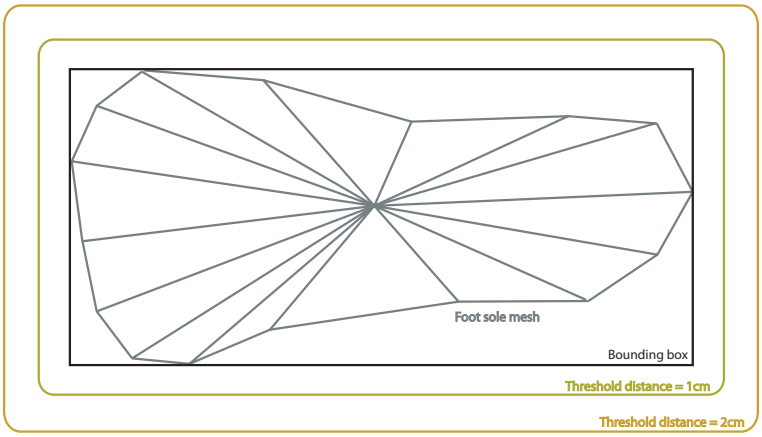

Figure 4.13: Illustration of the supporting area and threshold distances calculation. In grey is the foot sole mesh. First its bounding box is extracted so that it can be used as supporting area. From this bounding box, the acceptable area for the ZMP to lie in is calculated. The area depends on the maximal acceptable distance between the ZMP and the supporting area, as seen with the two example areas (in green and orange).

4.8 Multi-Dimensional Interpolation

Modifying the motion of the character takes some time due to the several non-linear optimizations that take place. Thus, to be used in an end user application, the adaptation

related to a specific set of sizes should be achieved at least in interactive time. To overcome this issue, we devised a framework that pre-calculates the corrections for a given number of examples shapes, and interpolates these at run time in order to quickly come up with the adequate corrections. To do so, we use scattered data interpolation techniques, which have proven to be efficient for this purpose.

4.8.1 Radial Basis Functions

Radial Basis Functions (RBFs) are a common way to interpolate an example data set in order to estimate the value of intermediate samples. The most common formulation of a RBF $\varphi : \mathbb{R}^n \rightarrow \mathbb{R}$ is:

$$\varphi(X) = \sum_{i=1}^{N} a_i \rho(\| X - C_i \|) \tag{4.14}$$

with N the number example data points, C_i the center vector of example data i, a_i is the weight associate with the example data i and ρ an interpolating function.

A better formulation, called normalized radial basis function (NRBF) is as follow:

$$\varphi(X) = \frac{\sum_{i=1}^{N} a_i \rho(\| X - C_i \|)}{\sum_{i=1}^{N} \rho(\| X - C_i \|)} \tag{4.15}$$

A theoretical justification of this formulation can be found in [BA03]. Intuitively, RBFs simply are a way to calculate a data sample by a weighted interpolation of existing data (figure 4.14). If the sum of the interpolated data is not equal to one, then the resulting data does not properly reflect the examples and should thus be normalized, hence the NRBF formulation.

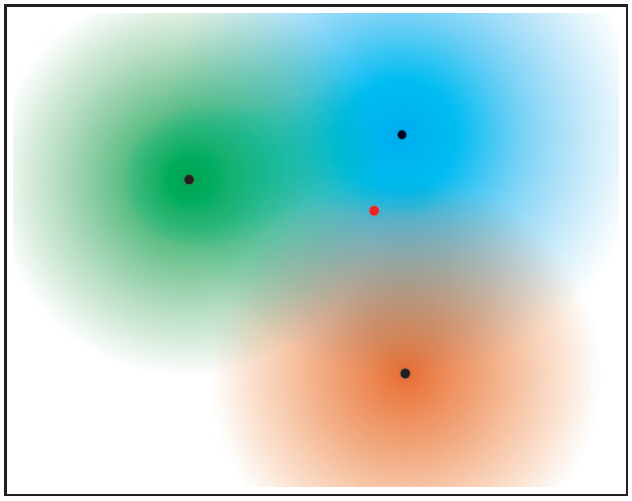

Figure 4.14: Conceptual principle of a radial basis function. Here in this 2D example, three sample data points (black dots) each have an influence over their neighboring regions (colored area), and a new point (red dot) can be interpolated by taking into account the influence and confidence of the example data.

The most commonly chosen function for ρ is to use a Gaussian:

$$\rho(\| X - C_i \|) = \exp[-\beta \| X - C_i \|^2] \tag{4.16}$$

Gaussians are considered local in the sense that their value decreases to zero when tending towards infinity:

$$\lim_{\|x\| \to \infty} \rho(\| X - C_i \|) = 0 \tag{4.17}$$

In our case, we have the possibility to choose where the example data point are taken in the data space. If we choose an adequate function for ρ then it is possible to accurately re-synthesize the existing data points. By adequate we mean that the function should truly be local and not just only tend towards zero. Moreover, it should have the nice interpolation properties of the Gaussian function so that the interpolation remains smooth.
The hanning function [PFTV92] is commonly used in signal processing [BT59] and appear to have all the desired properties:

$$h(t) = \begin{cases} 0.5 - 0.5\cos(2\pi\frac{t}{T}) & t \in [0, T] \\ 0 \quad \text{otherwise} \end{cases} \tag{4.18}$$

Its value really is zero outside the defined bounds, and its shape is almost the same as the Gaussian within the interpolation range (figure 4.15). By using this function in place of a Gaussian, we can truly limit the influence of the sample through the interpolation space.

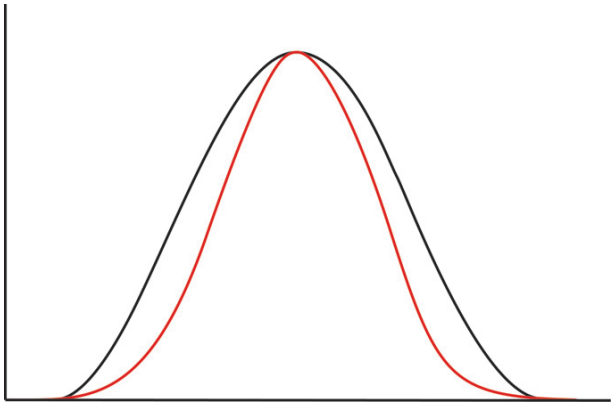

Figure 4.15: Comparison between Gauss (in red) and Hanning (in black) functions. the functions parameters are chosen so that the center and span of each function match.

Regarding the distance function, the Euclidian distance is the most commonly used one. However, its equidistant lines are circular and thus the influence of a particular example point cannot strictly be bound between square areas. Instead, we use the infinity norm (also known as Chebyshev distance) defined as follow:

$$d_{Cheb}(p, q) = \lim_{k \to \infty} (\sum_{i=1}^{n} |p_i - q_i|^k)^{1/k} \tag{4.19}$$

With p and q n-dimensional vectors.

It is easy to calculate this infinite norm: $d_{Cheb}(p,q) = \max_i(|p_i - q_i|)$. This distance much better corresponds to our needs because equidistant lines draw hypercubes in the parameter space instead of hyperspheres. Thus if the example data samples are regularly spaced in all the interpolation dimensions, we can smoothly interpolate between the samples and guarantee that the calculated data points accurately reflect the example data set (figure 4.16).

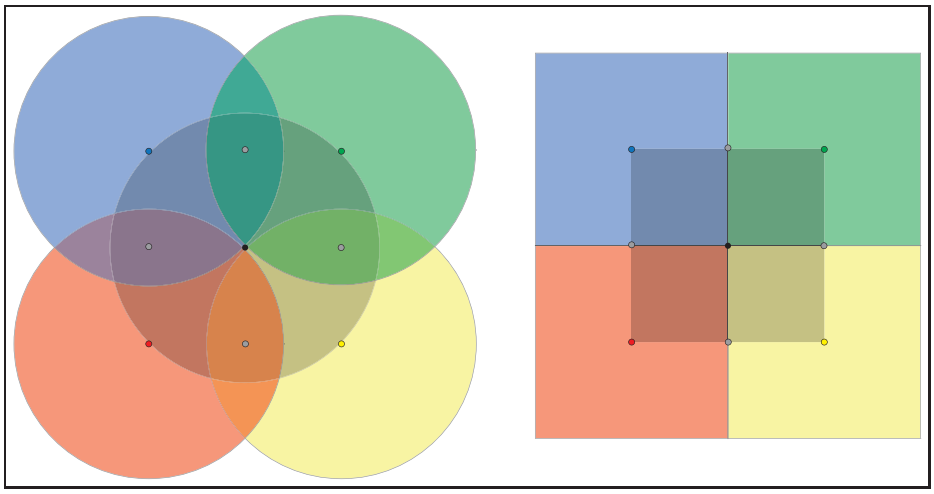

Figure 4.16: Comparison between the use of the Euclidian distance (left) and the infinity norm (right) in 2D. On the left, equidistant lines draw spheres while the infinity norm draws squares. If the sample data points are regularly spaced, this enables to accurately re-synthesize and interpolate the sample data set.

4.8.2 Data Pre-Processing

A customizable body model is composed of at least thirty segments which can be individually deformed. Even though the arms and legs adaptation only rely on the size of the arms and legs, the balance correction is dependant on the individual growth of each of the segments. To take into account all the subtle equilibrium change induced by each of the body segments, we would have to make each segment vary individually to cover the entire interpolation space. In that case the interpolation space would have thirty dimensions and if we want to calculate three states for each variables (small, neutral and big) then the number of optimizations to perform would be 3^{30}, obviously a number too big to be tractable in practice. Instead, we reduced the number of variables to five, namely the size of both legs, trunk and both arms (figure 4.17). This simplification still yields to good results in practice because as long as the global modification of the limb is taken into account the variation of individual segments within a particular limb has little influence on the final result.

Thus, each of the limbs is successively reduced and grown in diameter while keeping the other limbs size fixed. The pre-calculated example data can then be interpolated by RBF with the Hanning function and Chebyshev distance.

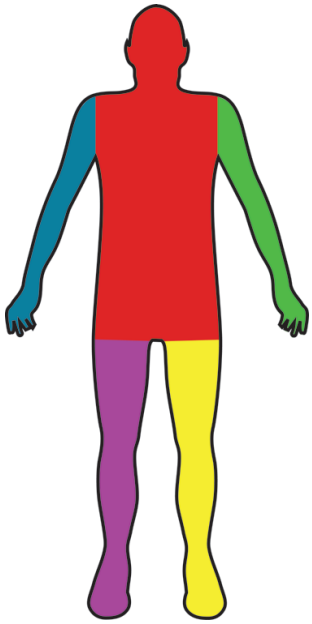

Figure 4.17: Split of the body parts for the balance correction. Five parts are retained, namely trunk (red), left arm (green), right arm (blue), left leg (yellow) and right leg (purple).

4.9 Skeleton Adaptation

We adapt the motion of a character to make its animation more suited to its actual shape. When an arbitrary character and animation are given to the system, there is no way to know whether the animation was already prepared by a designer or not. It is reasonable to assume that the skeleton provided with the body model and animation was designed specifically for that particular character, and hence should not be modified. In the case of a deformable body, the character shape and skeleton are provided in neutral configuration and can be modified to make it grown or shrink. As we know what the skeletal configuration for the standard shape is, the skeleton must be scaled along with the body growth, otherwise artifacts arise.

4.9.1 A Growing Body

There exist several ways to accurately deform a body shape so that it has specific dimensions. Seo [SMT03] proposed a method based on examples which are later interpolated with RBFs to produce new shapes with the desired features. This example based approach has one important drawback in the sense that because only examples are interpolated (mostly shapes from 3D scanners) then the resulting shapes resemble the input data and thus exhibits realistic features of the human body. Allen et Al. [ACP03] proposed a parametric approach to achieve the same goal. The way a body is deformed is defined by the user and can thus be bound to aesthetic shapes.
Both approaches require manual work in order to obtain visually pleasing shapes. Instead of using either one of these two we decided to propose a much simpler approach which even though it does not produce nice looking shapes is fast to implement and to use. Moreover, because we used this approach only for testing our motion adaptation

algorithms, we did not care so much about the visual aspect of the resulting bodies.
Our method relies on the skin attachment on the body along with the skeleton design in order to estimate how each vertex should be moved. To understand what the skin attachment means, one should know how the skin is deformed. We implemented the linear blend skinning method proposed by Magnenat-Thalmann et al. [MLT88]. This approach is still widely used for interactive applications and can be expressed as follow:

$$v'_i = \sum_j M_j B_j^{-1} v_i \quad j \in \mathcal{E}^i \tag{4.20}$$

with M_j the current transformation matrix of joint j, B_j the bind transform matrix of joint j, v_i the initial position of vertex i in bind pose, v'_i the deformed location of vertex i and $\mathcal{E}^i$ the set of influencing bones for vertex i.
Conceptually, each vertex is attached to one or more joints, with a weight associated to each influencing joint. The sum of the weights is equal to one, and they are carefully chosen so that the character smoothly deforms along with the skeleton motion. In order to be computationally efficient, the product $B_j^{-1} v_i$ from equation 4.20 is pre-calculated for each vertex, and thus each vertex is stored as a collection of offsets from their respective influencing bones, in local coordinates.
Additionally, the skeleton was done is such a way that the x axis of the bones are oriented towards the next joint. Considering that the limbs length do not change, then the growth of a limb is done by scaling the offsets in the y and z directions, as follow:

$$\begin{array}{c} W_i = \sum_j w_j \\ W'_i = \frac{\sum_k W_k}{K} \quad k \in \mathcal{K} \\ \left\{ \begin{array}{l} o_{ix} = o_{ix} \\ o_{iy} = o_{iy} + (\alpha - 1).o_{iy}.W'_i \\ o_{iz} = o_{iz} + (\alpha - 1).o_{iz}.W'_i \end{array} \right. \end{array} \tag{4.21}$$

with w_j the weight associated to the j^{th} influencing bone of vertex i, $\mathcal{K}$ the set of vertices directly connected to vertex i, α the scaling factor and o_i the offsets of vertex i. An example of such growth can be seen on figure 4.18.

4.9.2 Joints Translation

When the body grows, its joints should be resized accordingly. It is straightforward to estimate the scale that must be applied when the length of a segment is changed; however changing the girth of the trunk may also induce a skeleton resizing. Figure 4.2 shows that six segments connect the spine to the limbs, namely the clavicle, upperarm and thigh segments. Thus, when scaling the trunk by a factor α these segments should also be scaled by the same factor, as seen on figure 4.19. Depending on the magnitude of the scale, it might be possible to deal with changes only with the motion optimization stage. However, we noticed that we obtain much better results by also taking into account the resizing of the skeleton.

4.10 An IK Based Approach

The approach proposed in the previous section produced convincing results, however it suffers from one drawback: it is computationally expensive and thus we had to intro-

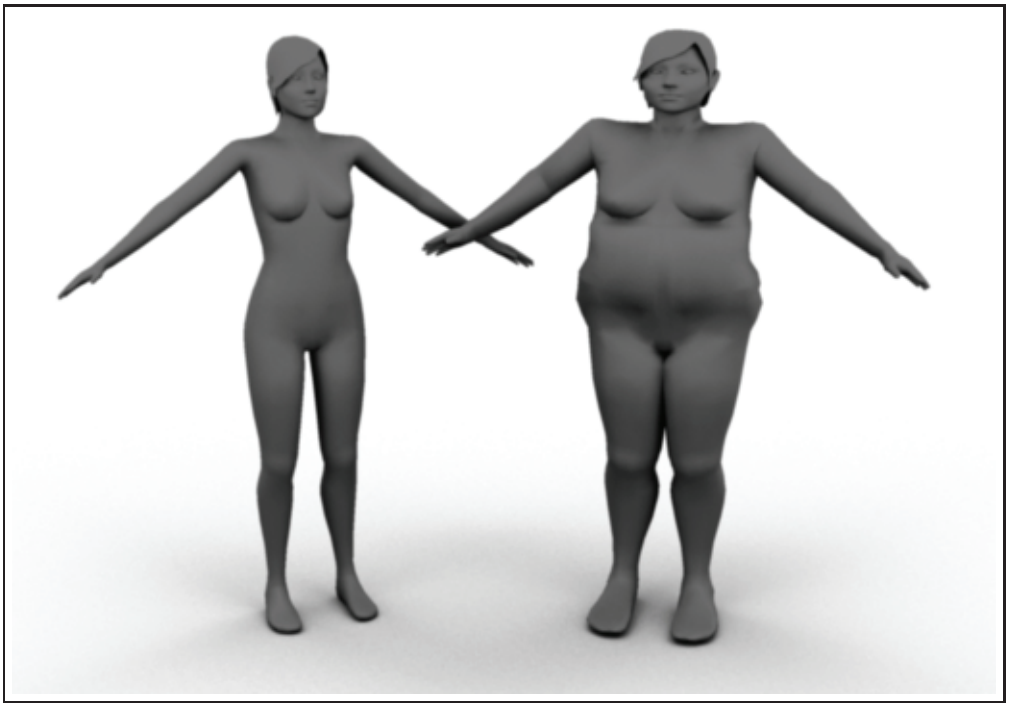

Figure 4.18: A skinned character (left) and its inflated counterpart (right) according to the method proposed in 4.9.1

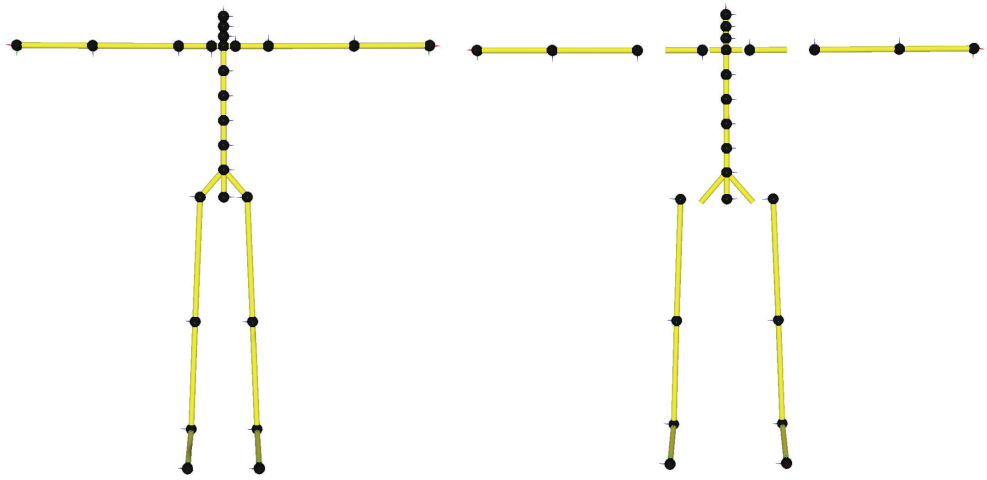

Figure 4.19: Scaling of the skeleton segments according to the growth imposed on the offsets. On the left is the original skeleton; while on the right is the scaled skeleton after an offsets growth by a factor of two. Notice the gap at the clavicle and thigh joints.

duce the interpolation scheme presented in section 4.8 to make it usable in real-time. We used global optimization because it ensures that the corrections are smoothly applied over the animation. Another approach would have been to use a per-frame IK algorithm to get rid of the penetrations and apply a smoothing algorithm over the computed correction values to ensure smoothness. We tried this idea with mild success, as explained in the results sections.

4.10.1 Penetration Correction

Once the penetration distance d is computed (cf section 4.2.3), the limbs must be moved apart from each other so that no more penetration remains. As depicted on figure 4.20 the situation is as follow: the green and violet cylinders are in penetration by an amount of d meters, and they must be separated by acting on the joint which rotates around C. If we simplify the cylinders interaction to a threshold distance, as in section 4.4.1 then it is straightforward to calculate the correction angle that must be applied. If we decided to keep the cylindrical model, the calculation is a bit more complex. The exact amount by which the green cylinder should be rotated around C so that no penetration remains is a non-linear function which depends on the length l, the radius r of the cylinder as well as the orientation of each cylinder with respect to the other one. We choose to adopt an analytical IK approach which gives an approximated value for the correction to be applied to C, and refine the estimation if necessary by performing several steps of the same algorithm.

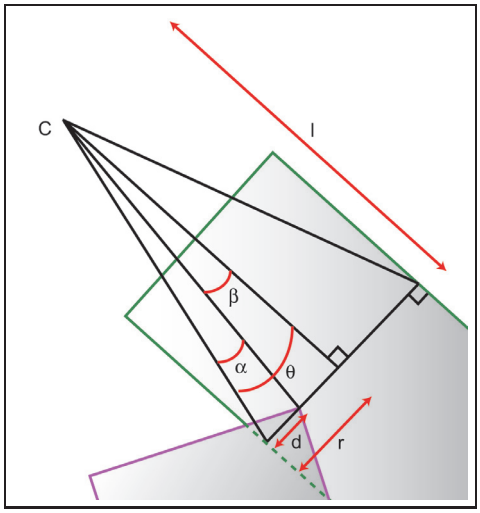

Figure 4.20: Typical case of two cylinders penetrating each others. In green and violet are two cylinders. d is the penetration distance, l is the height between the center of rotation and the line supporting d, C the center of rotation of the joint that holds the green cylinder, α,β,θ are angles that must be calculated.

The rotation correction α is calculated as follow:

$$\begin{array}{rcl} \alpha & = & \theta - \beta \\ \tan\theta & = & \frac{r}{l} \\ \tan\beta & = & \frac{r-d}{l} \\ \tan\alpha & = & \tan(\theta - \beta) \\ & = & \frac{tan\theta - \tan\beta}{1+\tan\theta\tan\beta} \end{array}$$

moreover, $\tan\theta$ and $\tan\beta$ are smaller than one, thus $\tan\theta\tan\beta \ll 1$ hence:

$$1 + \tan\theta\tan\beta \simeq 1$$

this yields to:

$$\begin{array}{rcl} \tan\alpha & = & \tan\theta - \tan\beta \\ \tan\alpha & = & \frac{d}{l} \end{array}$$

And eventually:

$$\alpha = \tan^{-1}(\tfrac{d}{l}) \tag{4.22}$$

Instead of this double approximation, we could have chosen to exactly compute the angle α. This can be done by calculating $\beta = \tan^{-1}(\frac{r}{l})$ and $\theta = \tan^{-1}(\frac{r-d}{l})$. However, α is not the exact value we are looking for either thus we have no guaranty that the exact value of α would be a better estimate of the correction than equation 4.22. Moreover, using equation 4.22 as a first guess for correcting the orientation of the colliding cylinder produces accurate results and it saves one $\tan^{-1}$ operation. It almost reaches the actual correction after a few iterations only, and practically only one iteration is enough to produce visually satisfying results.

For the arms, this correction is applied to the arm only as the torso is not likely to move due to the pressure exerted by the arms. However, for the legs, it makes sense to split the calculated correction in two and to apply half of it to both legs, and that is what was implemented. However, due to the geometric sharpness of the contacts between the cylinders, the corrections stem discontinuities regarding the smoothness of the resulting motion. The legs are indeed not penetrating each other any longer but due to the discretization of time over the recordings, the limbs tend to jump out of their original trajectories when a penetration used to take place. For addressing this secondary issue, we implemented a smoothing to the corrections, as outlined in the next section.

4.10.2 Smoothing

The algorithm described in the previous section produces a collection of corrections to be applied to the animation. However, as previously stressed, these corrections introduced discontinuities to the animation thus spawning unpleasant visual artifacts such as limbs jumping from one pose to another. Hence we now apply a smoothing algorithm corrections so that the resulting motion is both smoother and better matched to the original animation.
Even though we correct the animation data, we would like to do it as little a possible while ensuring that no penetration remains and that the resulting animation is still

close from the original animation. It is somewhat cumbersome to define what it really means to have two *similar* animations, as many criterion could be taken into account. Motion blending has introduced metrics for evaluating whether two motions are close from each other or not [KGP02, KG03, AF02, WB03] but this approach does not fit our problem here as these metrics rather define whether two motion are close or not than how to make them more similar.
There are two ways of thinking about metrics between motions. First, one can evaluate for each frame of the animation the spatial location of feature points disseminated throughout the body, and then simply add all the distances to get a scalar value which somehow express how far away the two clips are. Another way to think about it is to state that even if two characters are morphologically different, they can still perform the same action at different scale, and thus the measurement should take place at the joints orientation level rather than on the actual location of feature points.

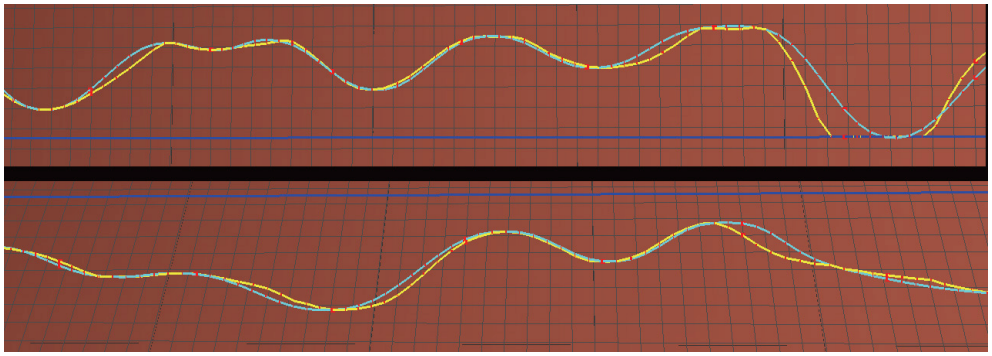

Figure 4.21: Two examples of smoothed corrections in 3D. On the horizontal axis is the animation time line, and on the vertical axis is the amplitude of the correction. The yellow lines are the raw corrections after their computation, and the blue line is the smooth approximation. The red dots are the animation frames.

We consider that even if two characters are morphologically different, they still can perform the same motion at different scale. Hence we start from a motion, apply corrections to the animation and would like to smooth the corrections so that the resulting motion is as close as possible from the previous one. One way to make the resulting motion look alike is to apply the corrections gradually so that the features of the original motion remain. For instance, if a correction is applied gradually over all the frames of the animation, then the limbs onto which it is applied slowly drifts away from its previous location, but it still moves the same way as the original motion. On figure 4.21, one can see two examples of such a smoothing which is performed using Sequin et Al. [SLY05] spline. This class of spline was chosen because it offers two relevant features. First, they go through the control points and not only approximate them. Second it offers C^2 continuity with minimal curvature, which is the feature we are looking after to ensure that the corrections are gradually applied.

As spline curves are meant to be applied to control points in an n-dimensional space, we must turn the correction values into these points. For doing so, we treat time like other spatial dimensions and each axis around which the corrections take place is allocated a separate dimension in the smoothing space. Because the corrections take place on two

axis - namely Y and Z, the resulting smoothing space has tree dimensions. For further simplicity, the unit of the dimension onto which time is mapped is not the *second* but instead the *animation frame*, and the correction angle values in radians are directly mapped onto the remaining dimensions of the smoothing space without any conversion. The curve itself is computed from the mapped control points using the following algorithm:

1: $AC \Leftarrow$ The corrections C smoothed by an average filter
2: $MinMax \Leftarrow$ All the local minimas and maximas of AC
3: **for all** Sets of four consecutive minimas and maximas p_i, p_j, p_k, p_l **do**
4: InterCurve$(p_j, p_k) \Leftarrow$ SmoothInterpolation(p_i, p_j, p_k, p_l)
5: **end for**
6: **for all** Sets of two control points p_i and p_j **do**
7: **if** $\exists c_k$ a raw correction and s_k a smoothed correction with $i < k < j$ so that $\|c_k\| > \|s_k\|$ and $\forall l \neq k, i < l < j$ then $\|c_k\| \geq \|c_l\|$ **then**
8: Add p_k as a control point
9: **end if**
10: **end for**
11: **for all** Sets of four consecutive minimas and maximas p_i, p_j, p_k, p_l **do**
12: InterCurve$(p_j, p_k) \Leftarrow$ SmoothInterpolation(p_i, p_j, p_k, p_l)
13: **end for**
14: $C \Leftarrow$ InterCurve

In the above algorithm, i, j, k, l are the frame indices of the control points. The first loop simply interpolates the minima and maxima. The second loop goes through all the corrections and smoothed values and check that indeed the curve was well approximated. If not, then it adds control points wherever needed. The third and last loop simply re-computes the smooth curve from the updated set of control vertices.
This algorithm gives good results in terms of approximation of the corrections, and no artifacts due to the corrections are visible afterward.

CHAPTER 5

Implementation and Results

5.1 Introduction

MIRALab has been involved in virtual human animation for more than a decade, and thus its know-how in the field became extensive over the years. The work presented in this thesis was integrated to MIRALab's animation framework so that it can be used and speed up the manual work that is still required to produce high quality animations.

Open Scene Graph (OSG) [web08b] is the 3D engine that is currently used by the lab to produce the real-time 3D content. OSG is an open source 3D graphics toolkit written fully in C++ and OpenGL. It was designed as an object oriented middleware, to make the development of graphics applications easier and faster. It provides scene graph management along with many libraries able to load and export various 3D formats. It also supports animation but soft deformations such as skinning are still missing in the toolkit. Because OSG is open source, it is possible to extend it to add missing features.

In the view of motion adaptation, we first had to extend OSG so that it can animate virtual humans by deforming their skin. Also, a suitable file format had to be picked so that models can be taken from CG software such as 3DS max to the OSG engine. For this, we picked the Collada format [com08] because it encapsulates all the data we need: models, textures, animations and skinning.

5.2 Architecture

OSG is organized around a scene graph architecture, with callbacks being made to appropriate functions while traversing the graph. The animation is achieved by one of these callbacks, which updates a given node according to the time elapsed since the last frame. Because motion adaptation is to be done once per animated model, we choose to create a separate class called `AnimOptimizer` with it own data structures (figure 5.1). This leaves the OSG callbacks and graph traversal untouched, hence ensuring maximal performances at runtime. After a character and animation were loaded through OSG, this data can be given to the `AnimOptimizer`, which then adapts the motion and eventually overrides the OSG animation data.

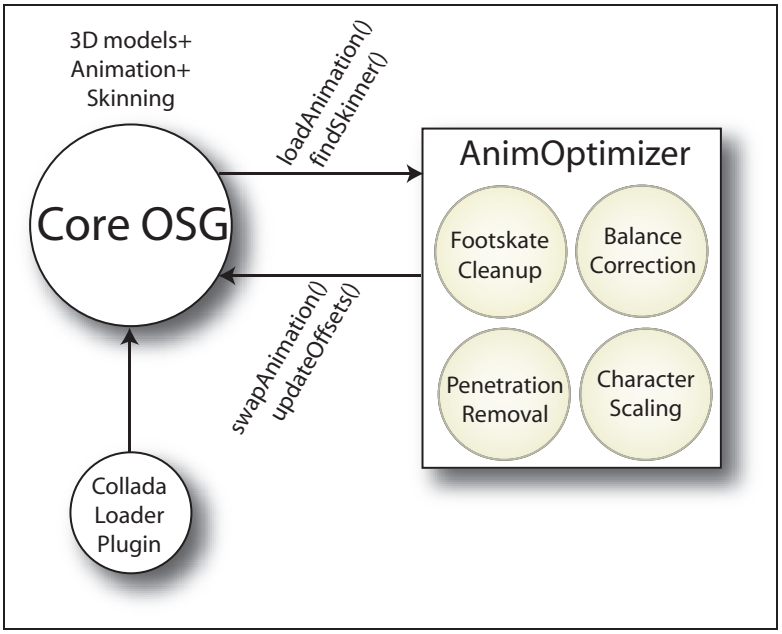

Figure 5.1: Architecture of our integration to OSG. The `AnimOptimizer` class loads the required data from OSG, which it can override after adaptation.

The `AnimOptimizer` stores internally a collection of collision volumes calculated from the character shape, and also a collection of animation frames (figure 5.2). The animation data can be altered by any of the adaptation modules independently depending on the requirements of the final animation. The arms penetration removal was split so that the left and right arms can be processed independently while the legs penetration was kept in a single method because of the inherent coupling between the two legs. However, even though each module can be used separately, a standard procedure can drawn because changes are applied only when needed, as follow:

- Create a new `AnimOptimizer` object.
- Load the animation data and 3D models.
- Find which vertices are part of the foot sole.
- Calculate/Load the vertex planting, i.e. which vertices of the foot soles should be planted and when.
- Calculate the mass of the different parts of the body.
- Calculate the per-frame threshold distance for the ZMP.
- Calculate where should the control points used for the optimization be placed.
- Remove the arms penetration.
- Remove the legs penetration.
- Remove the remaining penetrations.

- Remove the induced foot skating.
- Correct the balance.

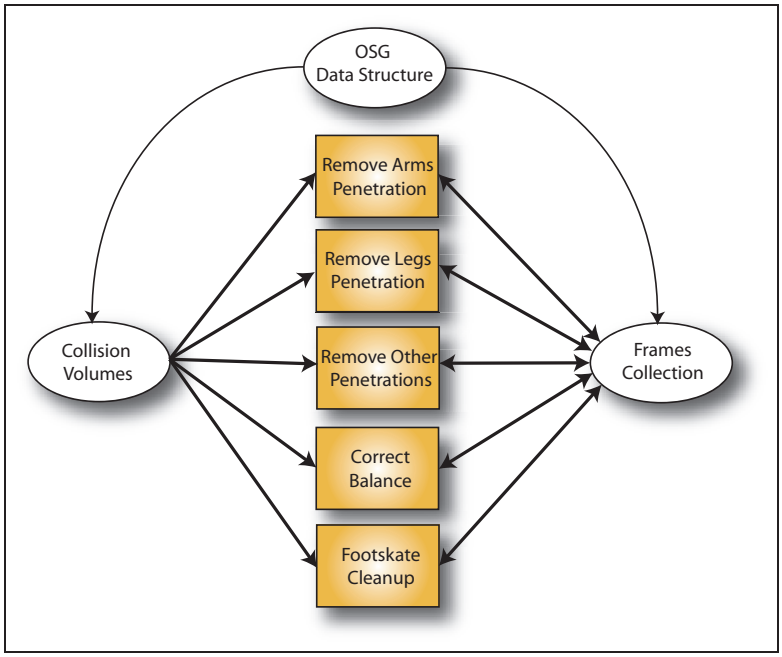

Figure 5.2: Processing of the animation data by the `AnimOptimizer` functionalities. Each can be applied separately as the same data structures are shared by all of them.

All these functionalities were implemented in the `AnimOptimizer` class, and can be used right away. However, OSG does not support skinning by default, thus the first piece of software we had to implement was a skin loader and deformer. The loader simply consists in retrieving the skinning data in the Collada file, while the deformer has to be optimized for OSG in order to maintain high performances at runtime.

5.3 A Simple OSG Skinner

We implemented a linear blend skinning [MLT88] algorithm so that the character's skin is deformed according to the motion of the skeleton, as said in section 4.9.1. Equation 4.20 tells us that the current global transforms of each joint is required to be able to deform the skin. OSG stores the data in a scene graph and thus all the transforms are relative to their parent node. Moreover, there could be more than one transform between 2 bones, and thus the relative transform between these 2 bones is the concatenation of all the relative transforms found between their 2 corresponding nodes in the scene graph (figure 5.3).

OSG provides a method called *computeLocalToWorldMatrix* which returns the world transform of a given node. We did not use it for two reasons. First, the transform that is returned is the transformation all the way up to the root of the scene graph. Second, using this method would mean that for each node, a full traversal of the scene graph is performed which is not optimized for real-time applications.

Instead we keep a pointer to each of the bone nodes, and perform the calculation ourselves in order to do as little computations as possible. Hence, for each of the bone node, we traverse the scene graph upwards, accumulating the transforms on our way, until we reach the father bone node of the animation root. This gives us the relative transforms between the bones, and thanks to an internally stored hierarchy we are then able to re-calculate the world transform of each bone with as little matrices multiplications as possible.

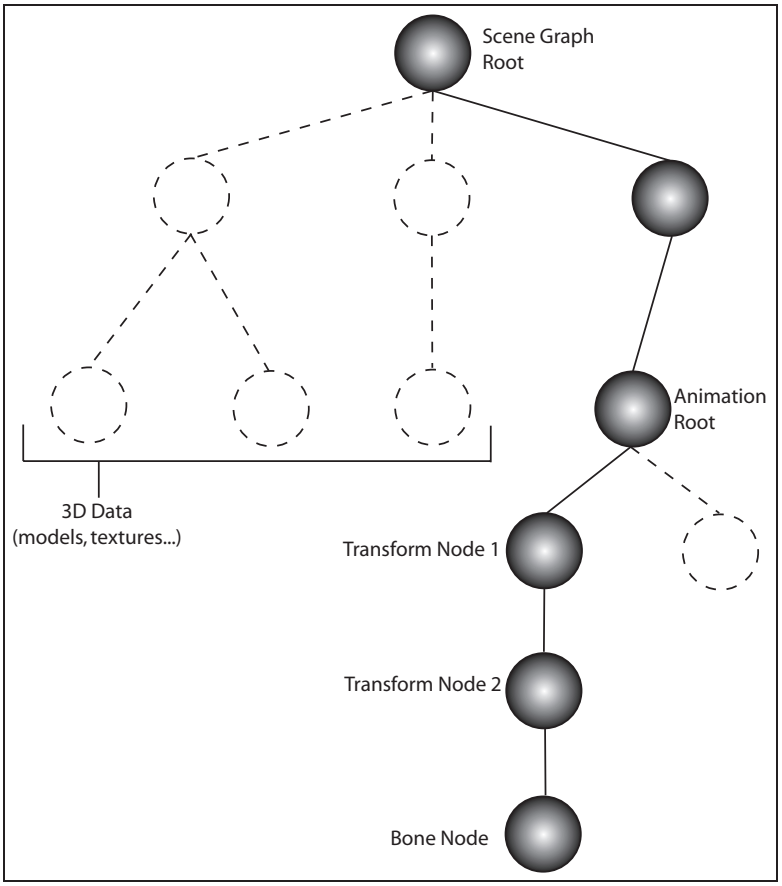

Figure 5.3:

Once the world transforms are calculated, it is then possible to deform the skin vertices according to equation 4.20. For the normals, it is widely known that these must be multiplied by the inverse transpose of the matrix used to transform the vertices. The reason for this being that given a vector T tangent to a polygon and a vector N the normal of this polygon, then their dot product should be null:

$$N.T = 0$$

After applying any transform to the mesh, these two vectors must remain perpendicular; otherwise the normals are not perpendicular to the mesh any longer. If we call the applied transformation matrix M, then T can safely be multiplied by M as it can be viewed as a difference between two points p and q, themselves transformed by M.
If we call the matrix used to transform the normal K, then the following relationship

must be verified:

$$KN.MT = 0$$

The dot product can be replaced by a vector product:

$$(KN)^T MT = 0$$

The transpose of a matrix multiplication is the multiplication of the transposes and thus:

$$N^T K^T MT = 0$$

Because $N.T = 0$, then $K^T M = Id$ and $K = M^{-T}$, the inverse transpose of the usual transformation matrix.
Vectors and normals are not influenced by a translation and thus the above result is only valid if M has a null translation. If not, then special care must be taken in order to reset the *w* component of the normals to zero after applying the inverse transpose matrix.
In the case of rotations only (i.e. when no scaling is involved), the inverse transpose is the rotation itself: $R^{-T} = R$. For body animations, no scaling is involved in the skeletal animation and thus it becomes useless to invert the transformation matrices. Instead, we simply isolated the rotational part of the transforms, and use these to deform the normals, thus avoiding to invert all the skeleton matrices at each step of an animation.

Eventually, this skinning class was implemented by extending the `NodeCallback` generic class. `NodeCallbacks` are meant to be applied when the `update()` method is invoked. Updates are performed before displaying every frame (e.g. for updating the animatio). Thus, by adding our skinning class to the stack of treatment to apply, we are assured that the skin is deformed according to the current animation pose.

5.3.1 The Collada Format

COLLADA (COLLAborative Design Activity) is an interchange file format for interactive 3D applications. Based on XML, this format is currently being developed to enable various graphics software applications to exchange digital assets otherwise stored in incompatible formats.
Originally created by Sony Computer Entertainment as the official format for the PlayStation 3, Collada is now the property of the Khronos Group. It is supported by the main player of the industry, such as Alias and Autodesk, plus many commercial game studios which adopted the standard for their file exchange needs. More recently, several 3D applications, such as Google Earth adopted Collada as their native file format for 3D content.

Among other features, a Collada file can contain mesh geometry, skinning, morphing, animation along with textures and CG shaders. The format itself only defines how the data should be stored, but not how it should be processed. For instance, for the skinning data, no guidelines are given regarding how the skin should be deformed from the skinning data contained in the file. It is thus left to the discretion of the implementer to choose the best way to combine the data to produce the desired result.

Structure

A Collada file is composed of at least three main parts:

- The *header* which provides the metadata associated with the file. For instance who created it, when and using which tool.
- One or more *libraries*. Each library stores the objects that compose a 3D scene. Several types of libraries are available: animations, images, materials, effects, geometries, controllers and visual scenes.
- One *scene* which is a pointer to the scene to be displayed from the libraries.

Thanks to the use of XML to support the format, it is not required that all the objects composing a 3D scene are stored in one single Collada file. Indeed, the objects and even the scene can be spread between several files, the links being resolved thanks to the XML middleware.

The skinning data is stored under a controller tag (the other possible controllers are morphers) and has the following elements:

- A *bind shape matrix* that must be applied to the bind shape so that its location in 3D matches the bind skeletal pose.
- A *list of joints names* which are the name of the joints composing the skeleton.
- A *bind pose*. The shape of the mesh in its initial pose before deformation.
- A list of *weights*. These are the numbers to be used as weighting coefficients during the deformation.
- A *mapping* between the weights, bones and vertices. This mapping defines, for each vertex to be deformed, the number of bones that influences it and which weights are to be used with which bone.

Even though a Collada file might look barbarian at a first glance (an example is given in annex D), it is well organized. All the *sources* are associated with 2 fields: an array and the technique that must be used in order to access it. The Collada DOM constructs the associated data structures so that it is possible to traverse the hierarchy when loading a file.
During the loading of a skinned model, we traverse the OSG hierarchy, looking for nodes called with the same names as the specified bones. Once found, a pointer to these nodes is kept so that their current transform can be retrieved at runtime in order to deform the skin appropriately. The blending weights are duplicated as many times as they are used, to speed up the computation. Eventually, as said previously, the bind pose is transformed into offsets from the influencing bones to decrease the number of matrix multiplication performed during every skin deformation.

5.4 The AnimOptimizer Class

As mentioned previously, this class works in parallel of OSG, by modifying an internal copy of the animation data loaded from OSG. The method `loadAnimation`

loads the current animation stored in the OSG hierarchy under the `root` node. Once the motion adaptation is done, it can be sent back to OSG by calling the method `swapAnimation`.

three main tasks can be assigned to `AnimOptimizer`:

- Get rid of the foot skating.
- Change the length of the character's limbs.
- Adapt the animation according to the current character shape.

Foot Skating Removal

The foot skating removal is performed using the method described in section 3.1. Once the character and animation are loaded in OSG, and after loading the animation in `AnimOptimizer`, calling the method `removeFootSkating` removes the foot skating. Because the vertices that must remain planted must have been estimated beforehand, `findFootSoleVertices` and `calculatePlantedVertices` must be called beforehand. The first one, as its name suggests, looks for the vertices belonging to the sole of the feet. The second one looks at the motion of the previously selected vertices and tags the least moving one of a particular frame as planted for this frame. In case the estimation does not give satisfactory results, the method can load the planting from a file instead of estimating it on the fly.
These methods could have been grouped in one single function; however we left them separately so that each of the processes taking place can be replaced by a new, more efficient one or even by an interactive process depending on the production needs.

Skeleton Resizing

In the context of deformable models, when the shape of a character changes, the underlying skeleton used to animate it must be modified accordingly. For instance, if the arms are made longer, then the skeleton should be scaled. This implies to re-calculate the offsets between the character's vertices and the skeleton bones, and also to scale the trajectory of the root node according to the legs scale.

Indeed, if the character scaling is uniform, then the foot skating can be removed by scaling the trajectory of the root node accordingly. The intuition behind this claim being that if the character's legs are made twice longer, then it travels twice as far. Thus, instead of always using the general purpose foot skating cleanup algorithm described previously, we also implemented this solution to let the choice to the designers regarding which method better suits their needs. `scaleCharacter` does all that was explained above, i.e. scale the skeleton, re-attach the vertices and scale the root trajectory.

The parameters of the function simply are the scaling coefficients that must be applied to the character parts, as detailed on figure 5.4. A value of 1 meaning that the current dimensions remains untouched, 0.5 would make the limb half the length it used to be, while 2 would make it twice longer.
Using this function implies that the character was deformed beforehand; otherwise the

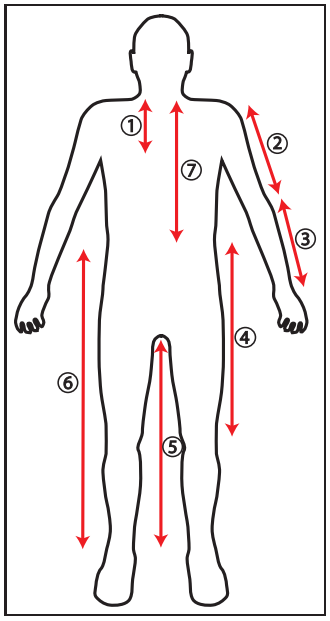

Figure 5.4: Scaling of the character supported by the `scaleCharacter` and `scaleSkeleton` method. 1: Scye length. 2: Upperarm length. 3: Forearm length. 4: Waist-Knee length 5:Inside legs length. 6: Waist Height. 7: Length of the back.

skeleton does not match the new shape. In case one would like to shorten or lengthen a body part without having to modify the character shape, then `scaleSkeleton` should be called instead.
It does take care of the new displacement induced by the new legs length; however the mesh does not get re-attached, which means that it will follow the skeleton scaling.

Gait Adaptation

For adapting the character's posture and gait according to its actual shape, several functions are available. We split them per limb, so that the individual control of what to correct is left to the user. `removeLeftArmPenetrations`, `removeRightArmPenetrations` and `removeLegsPenetrations` remove the penetration created by the character shape, while `removeCollision` takes care of the collisions created by the character's movements. This later function was left generic, i.e. the user must pass as an argument the colliding parts, along with the joints that can be modified to solve the issue. The balance correction is done through `optimizeBalance` in a fully automatic way.

Runtime System

The runtime system consists of a collection of files in which are contained the corrections associated with all the possible deformations of the character. The system loads the files and is then able to interpolate the corrections in real time. For this we implemented an *Interpolator* class. The files can be loaded using `loadInterpolationData` and interpolated using `interpolateCorrections`.

5.5 Results

5.5.1 Foot Skating Removal

Figure 5.5 exhibits the foot prints left by a retargeted walk (in blue) and by the original walk (in green). One can see that nearly all the foot skate is corrected. The minor sliding which remains is due to the estimation of the displacement of the root joint between two steps, as described in section 3.1.2. Indeed, because the weight transfer does not occur exactly on an animation frame, the feet may slide a little bit between two frames when switching from a foot plant to the other. Figure 5.7 shows animation snapshots of an original animation and its retargeted version. One can see that the drifting effect for getting rid of the foot sliding actually occurs, while the height of the character is modified so that its feet stick to the floor as much as possible. Figure 5.6 shows 2 other examples of adaptation. There again, no more foot skating remains after the adaptation of the animation, and the character's path was modified accordingly. This approach is fast and efficient, as long as the character still has at least one of its feet on the ground.

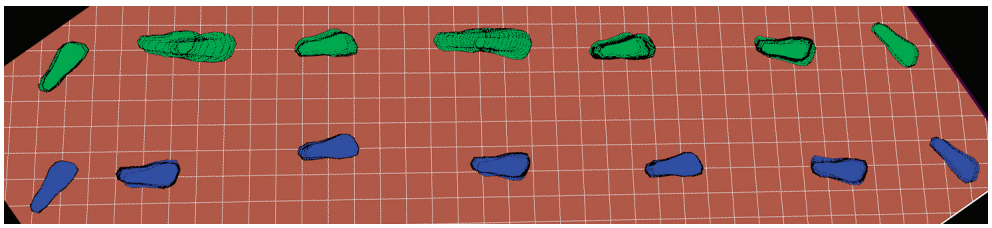

Figure 5.5: Two foot prints left by a character animation: on the left (in green) the original clip and on the right (in blue) the retargeted clip. The residual sliding that one may notice on the blue prints is due to the estimation of the root motion between two steps which makes both feet move at the same time in order to get a more realistic final motion.

Discussion

We presented a method to remove the foot skating of a motion clip which takes into account the character itself in order to improve the results and speed up the computations. Compared to previous approaches, this method fully relies on the skin to perform the retarget. It can be utilized by casual users as no interaction or expertise is required to obtain good motions. The foot skate removal preserves the original movements of the limbs because the corrections are applied to the root joint translation only. Moreover -for most cases- it has the advantage of extracting the foot plants automatically, which prevents time consuming manual work.

The tests we conducted with the IK based approach were not conclusive. Indeed, as pointed out by Kovar [KSG02], changing the height of the feet by a small factor introduces a noticeable bending of the leg. Even when spreading the corrections as far out in the animation as possible, the small, swift bending remained noticeable. One possibility to cope with this issue is to lift the ground (or lower the character) by a small amount. That way, in the original animation, the feet are more penetrating the ground that flying above it. This makes the legs bend a little bit more rather than stretch, which

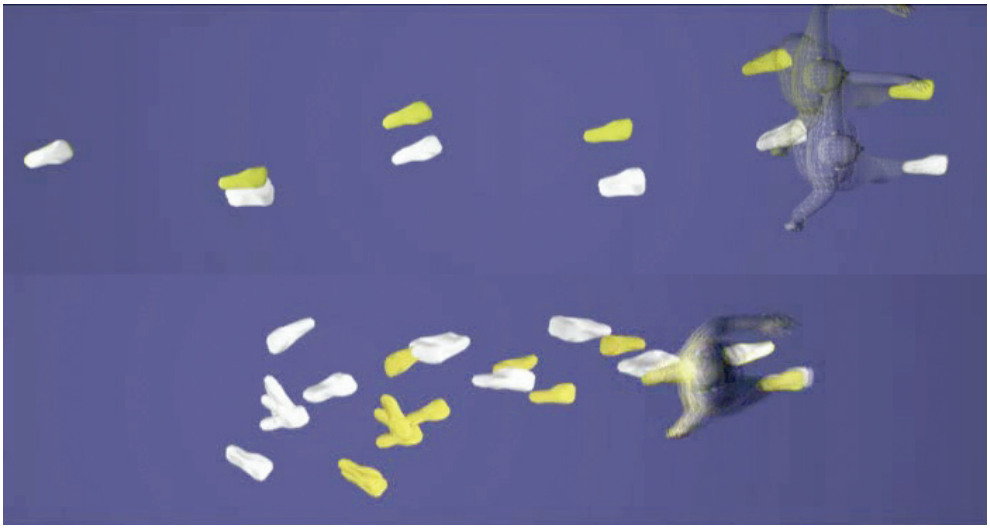

Figure 5.6: Foot prints left by two characters animations (top and bottom). The original prints are in white, while the corrected ones are in yellow. No more foot skating remains, and the trajectory of the character was adapted accordingly.

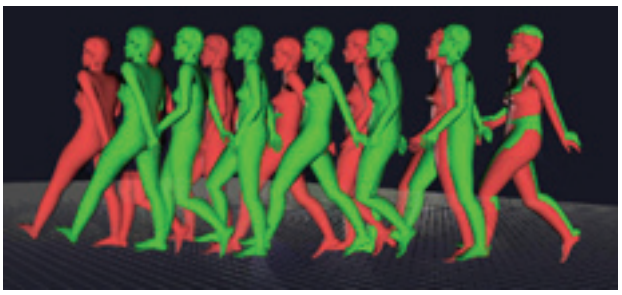

Figure 5.7: Superimposed snapshots of a walk with the original animation in red, and the corrected one in green.

seemed to be less noticeable.

Another advantage of the first method we presented over the second one is that if the character is arbitrarily scaled, then the first method takes that scaling into account automatically. The second method, however, requires that the global height of the character is already optimized for the motion being processed. Indeed, if the character is too high then the legs would be completely stretched, while if it is too low it would sneak rather than walk.

5.5.2 Character Based Adaptation

We implemented the motion adaptation algorithms in C++ using the RFSQP optimizer from AEM Design [LZT]. The motion adaptation itself took between 0.5 and one second per frame on a pentium IV 3.0 Ghz processor.
We tried our approach on various character shapes and animations and we successfully eliminated the self penetrations while maintaining a high level of realism. Figure 5.8

shows a the landing of a jump sequence. On figure 5.9 the posture was adapted to maintain the balance. Figure 5.10 shows the result of a penetration removal, while figure 5.11 is an full adaptation of a scanned body. For these examples the full adaptation has been applied, with noticeable differences emerging only when necessary. On the left, in red is the original posture, in the middle in green is the adapted one and on the right both postures are superimposed. The run-time system was tested by making each of the limbs successively grow and shrink in diameter while keeping the size of the other limbs fixed. We kept five dimensions for the interpolation space (legs, arms and trunk) and thus the number of optimizations to perform was $3^5 = 243$, which took approximately one days for an eight hundred frames animation. Snapshots of interpolated animations can be seen on figure 5.12. On the left is the original body shape, while the middle and right bodies had some of their limbs grown in diameter and their posture adapted.

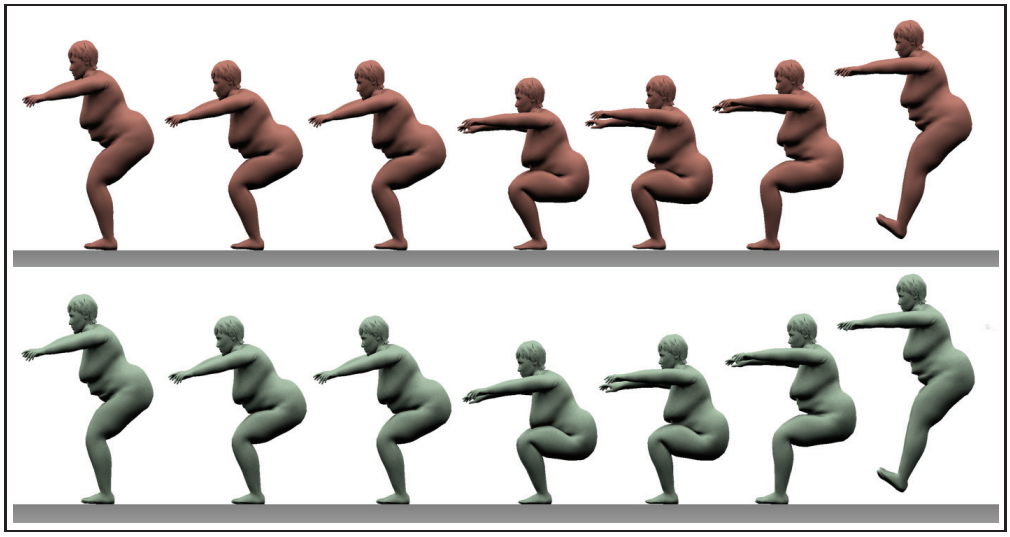

Figure 5.8: Jump motion. Top: adapted, bottom: original.

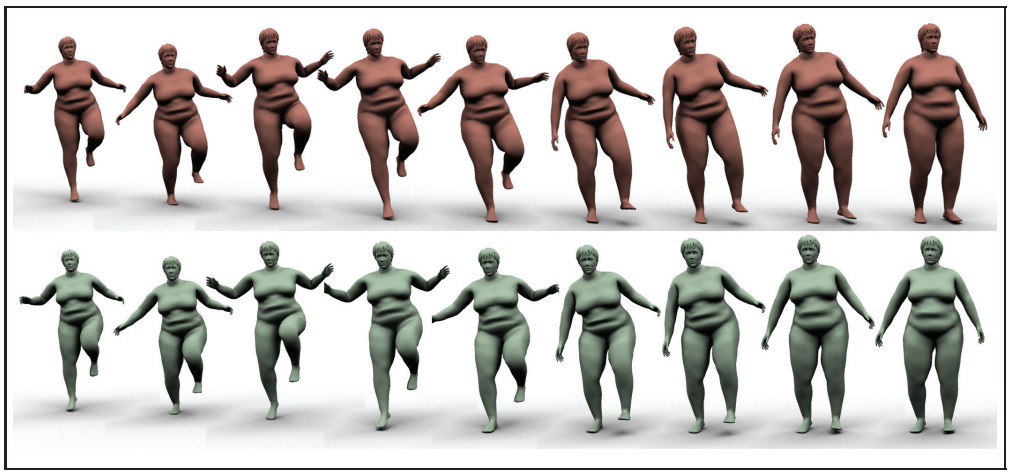

Figure 5.9: Skip motion. Top: adapted, bottom: original.

Figure 5.13 shows snapshots of the animation of a growing body, with its cloth

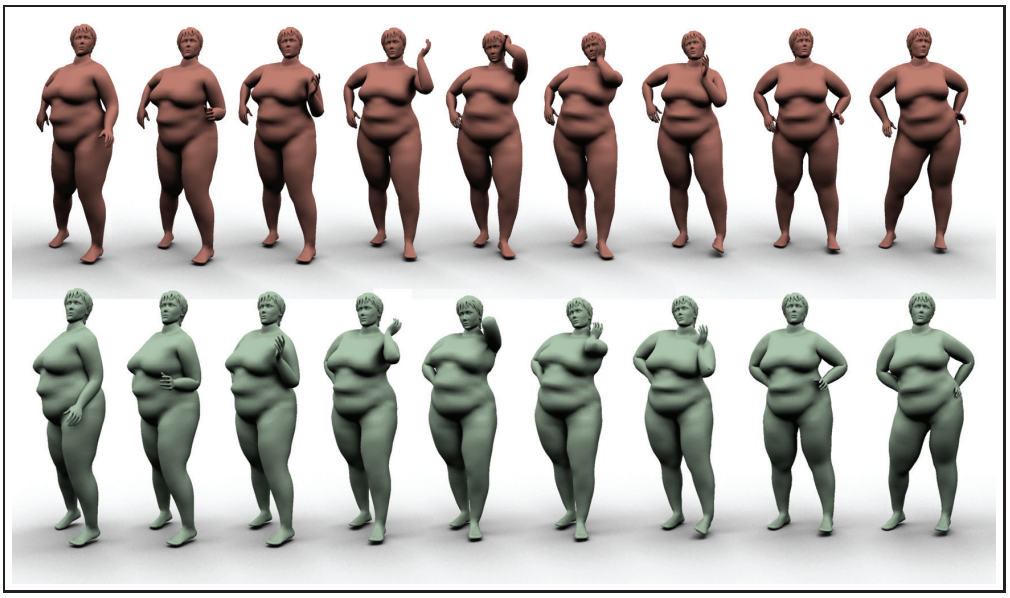

Figure 5.10: A character posing with self-collisions. Top: adapted, bottom: original.

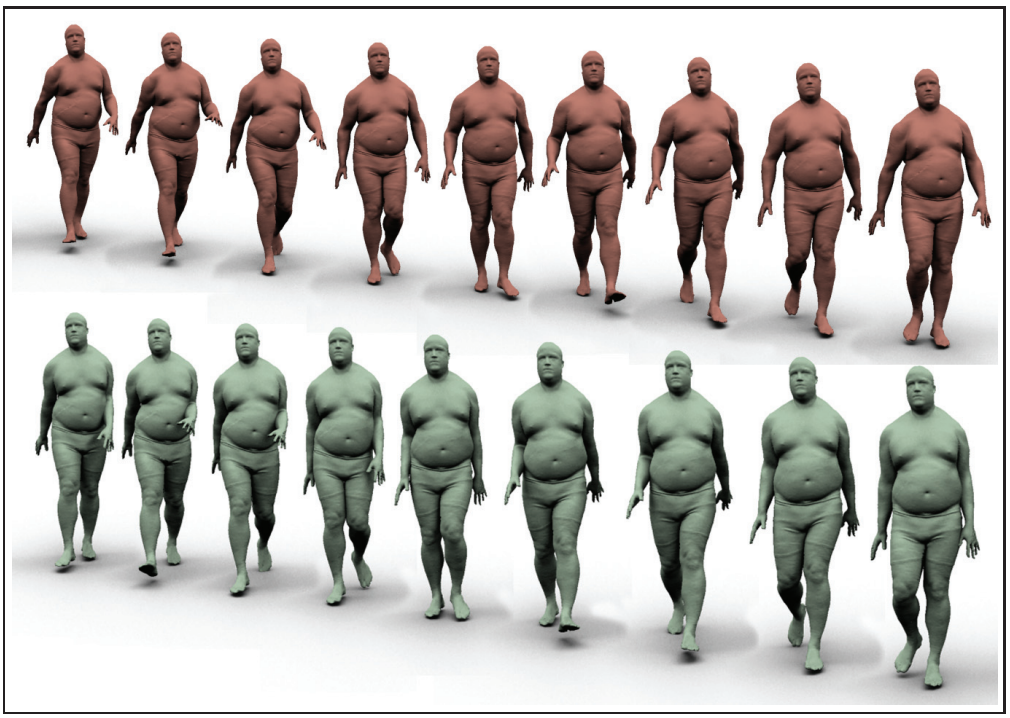

Figure 5.11: Walking character. Top: adapted, bottom: original.

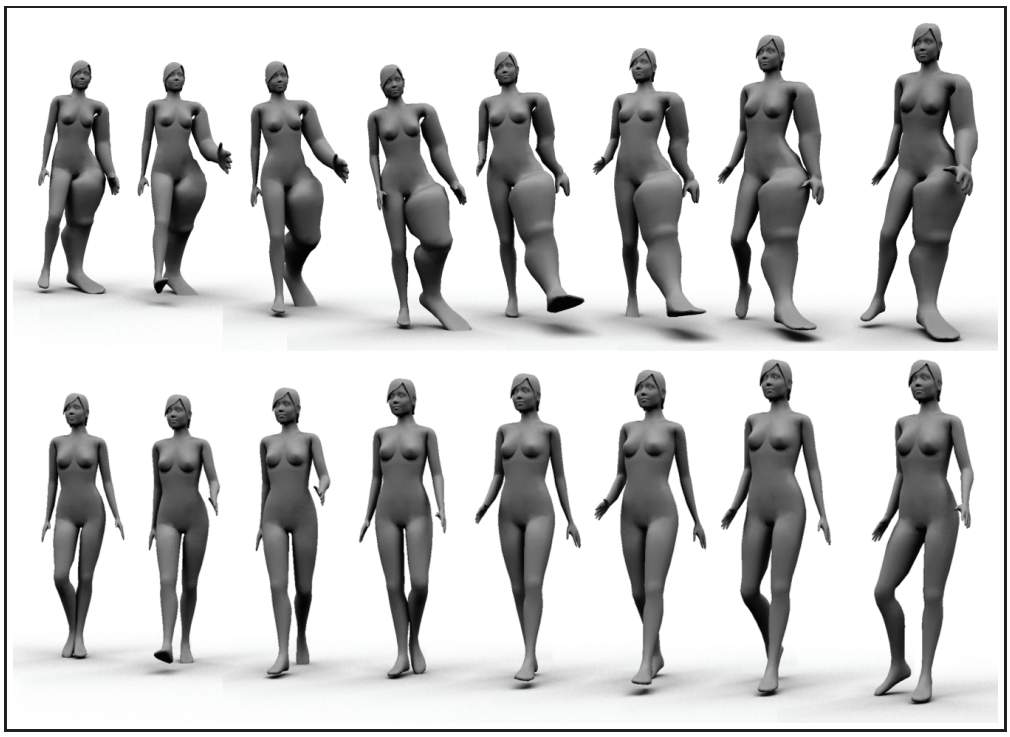

Figure 5.12: grown character. top: grown and adapted, bottom: original.

getting tighter over time. For each frame of this animation, the data had to be re-interpolated, which took a negligible amount of time and thus makes it usable in real-time applications like a virtual try-on for instance.

Figure 5.13: Snapshots of a walking sequence of a deformable character growing along time. The adaptation data was pre-calculated and interpolated at runtime to adapt the animation according to the growth of the character.

The analytic IK approach has proven to be efficient as well, as shown on figures 5.14 and 5.15. However, this approach is not as straightforward to use as the global optimization one. Indeed, Because the solutions are found per frame and then smoothed over the animation, a *bad* solution is kept by the adaptation framework and spread to the neighboring frames. With the global optimization approach, a *bad* solution is less likely to emerge because the magnitude of the corrections is taken into account during the adaptation process.

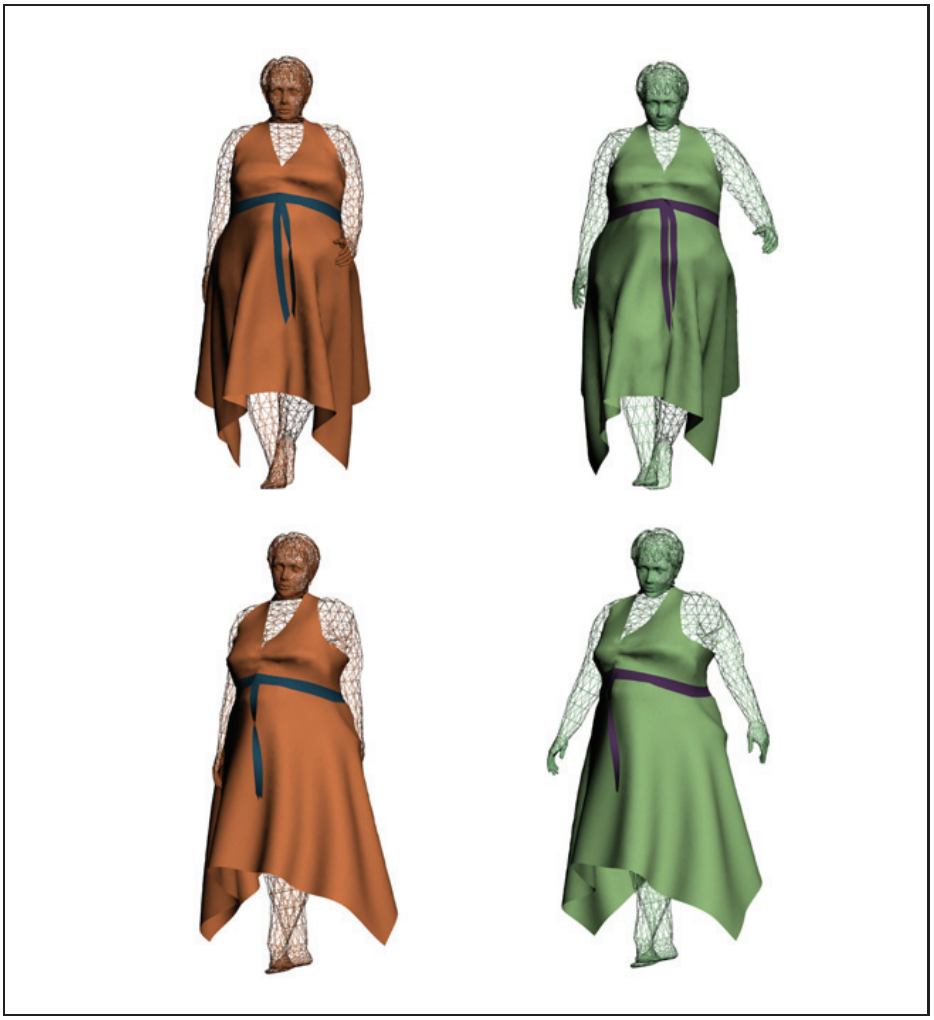

Figure 5.14: Snapshots of the animation of a plump lady. On the right are original postures of the character, while on the right the pose has been adapted.

Moreover, this approach does not take into account the balance of the character and thus the results are a bit less realistic than with the global optimization approach. The advantage of the IK compared with the global optimization is that it performs much faster, and thus a result can be tweaked by successive adaptations with different parameters.

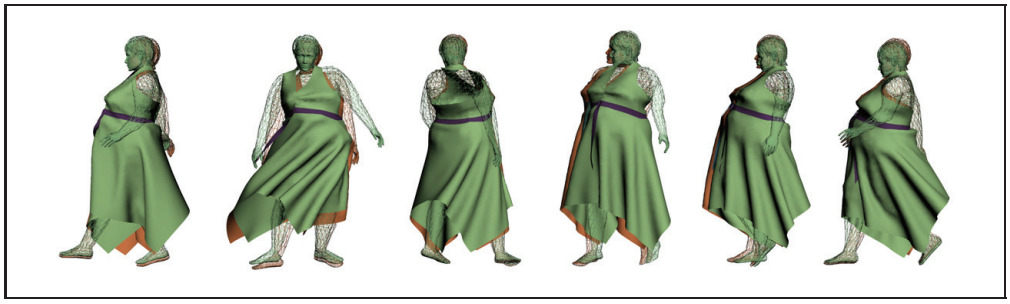

Figure 5.15: Snapshot of a walking animation of a plump character. In green are the adapted postures, while in red are the original ones.

Discussion

We presented 2 methods that are able to adapt a given animation clip so that the motion fits to any kind of body. The proposed approaches use either spacetime optimization or analytical IK. We also proposed a variant of the NRBF interpolation to make the corrections computable in real-time.
We recommend using the spacetime approach as we noticed that the results are of higher quality than with the IK approach. The explanation of this could be that the spacetime algorithm uses control points to change the character pose, which has the effect to directly smooth the corrections over the animation.

For the spacetime approach, we did not apply the forearm orientation correction to the hand because it generated too much correction, producing unrealistic postures. When the legs motion is adapted, the hip rotation makes the entire upper body rotate as well. In case one would like to keep the original head orientation (for controlling the gaze for instance), then the head joint can be rotated with the inverse of the hip rotation. Similarly, when the character adapts its posture the forearm orientation is also changed. This can be adjusted by applying the opposite rotation value from the spine to the shoulder. However we recommend being careful with this as it may introduce new penetrations which did not exist before.

5.6 Performances Evaluation

As stressed in chapter 2, our goal was not to transform the movements of a given subject into movements that someone else would have done. Rather, we animate very different body with a given motion clip. Thus, we know in advance that even if applied to someone's body shape, a motion clip will not be transformed into the movements that

this particular person would have done. This prevent us from comparing a transformed motion with a similar performance done by the target body's owner. Instead, we will evaluate to which extent the correction of the character's balance has or hasn't improved the overall stability of the character.

5.6.1 Evaluation Paradigm

We will first evaluate the accuracy of the cylindrical approximation we used to estimate the mass distribution over the body. To do so, we will compare the performances of the cylindrical model of the body with two other approximations: the fitting of ellipsoids to the body instead of cylinders, and the optimized mass distribution proposed by [SKG03]. These three ways of estimating the mass distribution will be weighted against real data, by calculating ZMP trajectory for several motions and for each method of approximation, and calculate the mean distance of each trajectory with the real trajectory: the one with the smallest average distance is the best approximation of the mass distribution.

The second kind of evaluation that we will perform is to evaluate if the balance correction really improves the stability of the character. As we took a threshold distance from the supporting area as criterion to estimate the balance of a motion, we look whether the ZMP is really brought back within this threshold distance or not.

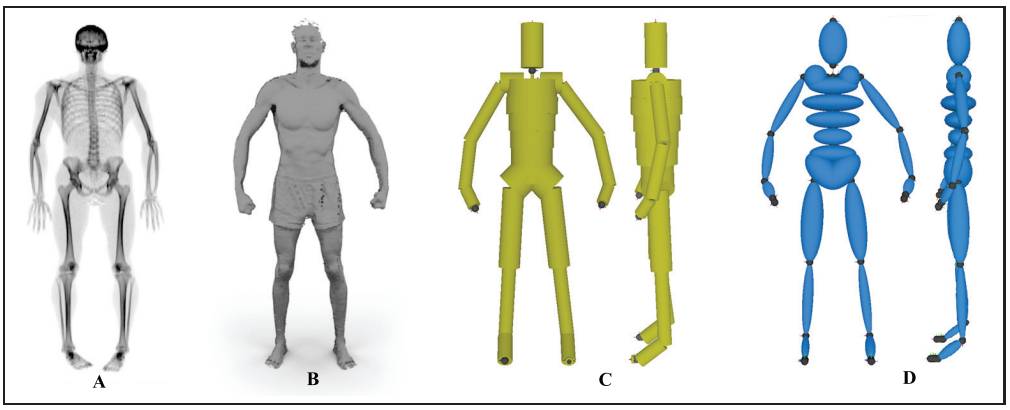

Figure 5.16: Input and output data of our performance evaluation. A. X-Ray view of my body. B. body scan. C. Cylindrical model D. Ellipsoidal model.

The real data was obtained by measuring the mass distribution, body shape and motion on the same subject. The mass distribution was measured through X-ray absorptiometry [SLW+06] which is a technique primarily used for measuring the bone's density. It also provides the mass and fat percentage of each limb, which is of great interest here. Because we only had the weight of each limb and not of the individual segments of the body, we had to estimate the mass distribution within each segment. For doing so we allocated *within each limb* the average of the mass distribution provided by the cylindrical and ellipsoid models. We could have reduced each limb to a single point mass instead, but it seems to us that this would have been less accurate, even though we agree that we introduced a small error here.

The body shape was acquired using a 3D body scanner manufactured by Human Solutions GmbH, while the motion was captured on a Vicon MX13 system at 100Hz and down sampled at 25Hz. We tagged the foot planting by hand, and approximated the shape of the foot sole by its oriented bounding box (OBB). In case both feet are planted, we took the supporting area as the convex hull of both feet OBBs.

5.6.2 Results and Discussion

The subject we captured was 29 years old, 184.8cm tall and weighted 63.98 Kg. The mass of its limbs calculated from picture 5.16A is reproduced on table 1 along with the masses obtained from the body scan (fig. 5.16B) by the cylinders (fig. 5.16C) and ellipsoid (fig. 5.16D) fitting.

Limb \ Model	Cylinders	Ellipsoids	Measured
Head	5043	2498	4690
Right Arm	3981	1340	4846
Left Arm	4325	1393	4570
Trunk	19772	18794	29955
Left Leg	8027	4579	9766
Right Leg	8077	3995	10154
Total	49227	32599	63982

Table 5.1: Total mass per limb and per model, in grammes.

As the actual mass of a limb is not as relevant as the mass distribution over the body (indeed, the result of equation 4.13 does not change if all the weights are scaled uniformely), table 5.6.2 shows the percentage of the total mass for each limb. The cylindrical model seems to be closer from reality than the ellipsoids, which is to be confirmed by the ZMP trajectories.

We calculated these trajectories for eight motion clips: normal walk, slow walk, fast walk, lean forward, run in circles, skip, jumps and 360^o rotation. We optimized the mass of the avatar for each motion using the RFSQP optimizer from AEM design [LZT] and reproduced the average mass and standard deviation (SD) on table 5.6.2. As one can see the masses are somewhat unrealistic (the legs are almost as heavy as the torso), and the standard deviations are often greater than the values themselves. This makes sense as we calculated the mass distribution to bring the ZMP as close as possible from the supporting area. Hence, the mass was relocated towards the feet in case of a very dynamic motion (for which the feet were in contact with the ground only during a short instants) and moved towards the center of mass in case of very static ones. The masses we obtained per motion are reproduced in appendix F.

Pieces of the ZMP trajectories we obtained for each model are reproduced on figure 5.17 and 5.18 while the average distances with the trajectory calculated from the real data are shown on table 5.6.2. The average distances per motion are given in appendix F. This confirmed the good results of the cylinders model as a good approximation of the mass distribution over the body. The ellipsoids performed well also, but their

Limb \ Model	Cylinders	Ellipsoids	Measured
Head	10.25	7.66	7.33
Right Arm	8.09	4.11	7.75
Left Arm	8.79	4.27	7.14
Trunk	40.17	57.65	46.82
Left Leg	16.31	14.05	15.26
Right Leg	16.41	12.26	15.87

Table 5.2: Mass distribution per limb and per model, in percent of the total mass.

Limb	Average Weight	SD
Head	999	1915
Right Arm	6246	6736
Left Arm	6960	5786
Trunk	18787	19942
Left Leg	18994	14752
Right Leg	11760	12414

Table 5.3: Average mass of the limbs calculated through optimization, in grammes.

associated ZMP trajectories remained a bit further from the real ones (except for the run in circle). Depending on the application, it may thus be possible to use either the cylinders or ellipsoids model.

We propose to apply the scaling reproduced on table 5.6.2 to the radius of the calculated volumes, so that the mass distribution better fit to reality. This non-uniform scaling would make the computed volumes reproduce exactly the mass distribution measured on our real subject (but not the real weights !). Even if we have no guarantee that these scaling would be the optimal values for other models, our experiments showed that a small change in the mass distribution does not influence the ZMP trajectory much. Thus we believe that the proposed scaling would improve the accuracy of the estimate.

Model	Average distance	SD
Cylinders	1.42	0.90
Ellipsoids	1.59	0.96
Optimization	9.08	4.93

Table 5.4: Average distance of the ZMP from the ground truth trajectory, in centimeters.

Limb	Cylinders scale	Ellipsoids scale
Head	0.85	0.98
Right Arm	0.97	1.36
Left Arm	0.90	1.29
Trunk	1.08	0.90
Left Leg	0.97	1.04
Right Leg	0.98	1.14

Table 5.5: Proposed scaling of the parametric volumes radius.

This experiment also confirmed that the ZMP does not always stay within the supporting area. Over the eight motions, and for the real weights of the limbs it remained at an average 3.68cm from the supporting area (the average distance per motion and standard deviations are given in appendix F).

For the balance correction, we looked at several original and corrected motion clips. The threshold distance is given by the original clip while the new ZMP trajectory is provided by the adapted clip, without applying the balance correction. Eventually, the final ZMP trajectory is obtained after applying the balance correction step. The values indeed decreased while the threshold distances were matched within a small distance, as seen on table 5.6.2. Examples 3 and 5 feature a final distance which is much closer than the original clip's distance. The reason for this is that we applied a scaling coheficient on the threshold distances in order to make the model more balanced than it used to be. This worked well for the skip motion, but for the walk it made the animation swing too much from left to right, thus leading to an unrealistic final animation.

Figure 5.17: ZMP trajectories for various movements. Top: Jumps. Middle/right: run in circle. Bottom/left: 360^o rotation. The trajectory calculated from the real weights is in black, the ones from the cylinders, ellipsoids and optimization are in blue, green and red respectively. The scale of the picture changes from trajectory to trajectory, but the size of one foot print (in grey) is always 29.6cm by 13.5cm.

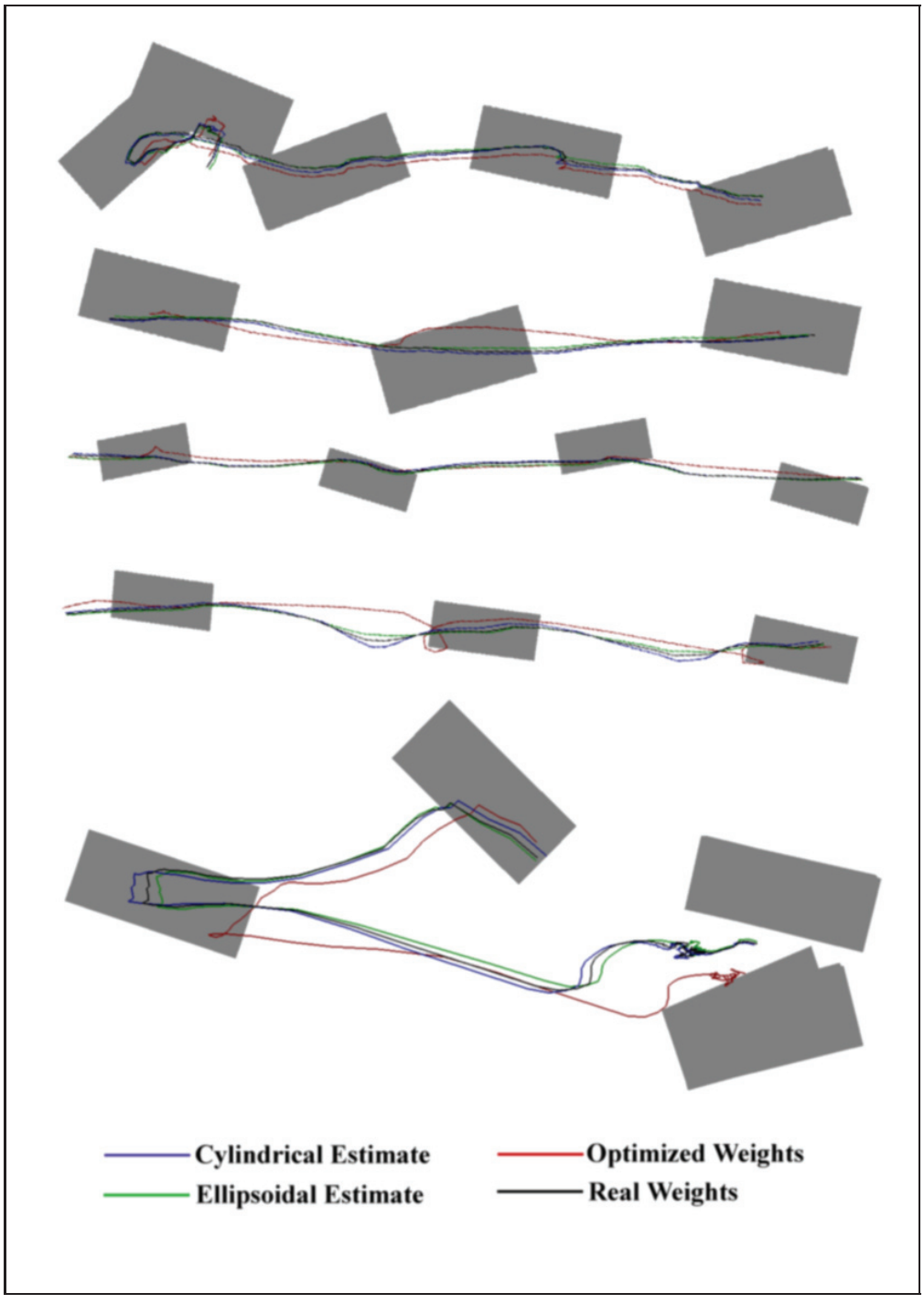

Figure 5.18: ZMP trajectories for various movements. From top to bottom: Slow walk, normal walk, fast walk, skip and lean. The trajectory calculated from the real weights is in black, the ones from the cylinders, ellipsoids and optimization are in blue, green and red respectively. The scale of the picture changes from trajectory to trajectory, but the size of one foot print (in grey) is always 29.6cm by 13.5cm.

The original distances were not matched exactly, even for the other examples (figure 5.19). The reason for this is that because we used control points to interface the motion curves and the optimizer, a given change is always applied to several consecutive frames of the animation. Thus, unless the control points are perfectly placed, the optimizer must find a compromise between the location of the ZMP and the magnitude of the changed introduced by the adaptation.

Example Clip	Original	Corrected	Balanced
1	8.38 (7.78)	10.06 (7.45)	7.35 (7.46)
2	11.27 (3.66)	11.31 (3.64)	11.17 (3.61)
3	6.60 (7.28)	6.61 (7.31)	3.72 (4.51)
4	4.71 (6.14)	5.54 (6.19)	4.73 (6.44)
5	5.11 (7.64)	5.14 (7.64)	4.49 (7.06)

Table 5.6: Mean distance of the ZMP from the supporting area, for the original, corrected and balanced clips. 1. Walk. 2. Jump. 3. Skip. 4. Another Walk. 5. Scanned body walk. In parenthesis is the standard deviation.

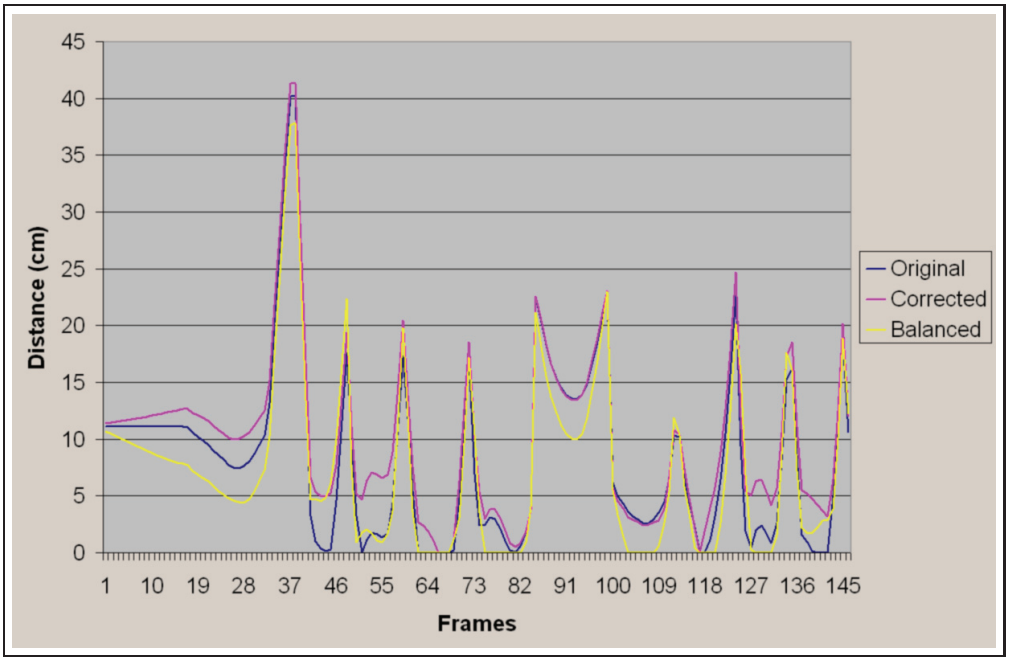

Figure 5.19: Plotting of the distances from the supporting area for a walk example.

CHAPTER 6

Conclusion

In this chapter we summarize what we presented to adapt a motion clip on any character shape. We will first review what we proposed, and in the last section we will discuss what we intend to improve in the future.

6.1 Contributions

As stressed in section 2.8, our primary goal was to be able to remove the self-collisions occurring when animating a virtual character. We have introduced a specific strategy for doing so in the case of the collisions created by the character's shape. The arms penetration removal keeps the original orientation of the forearm, while for the legs the hip is rotated to compensate for the large thighs. This had never been done before, and it allows to animated any character shape with a given motion clip.

Our secondary goal was to keep the balance of the motion at least as balanced as it used to be. As the test beds from section 5.6 demonstrated, we achieve to do it using spacetime optimization. We also introduced a new way to estimate the balance of a motion. Unlike previous approaches which assumed that the ZMP must lie within the supporting area, our experiments demonstrated that this is not always the case, and the a per-frame threshold distance should be used.

We also proposed a new way to remove the foot skating which is completely based on the character's shape. Instead of modifying the animation of the limbs, we change the trajectory of the root joint so that the foot skating is no more. The proposed algorithm also handles the possible penetrations in the ground by minimizing the height of the part of the foot in contact with the floor.

Eventually, to make the system usable in real-time we proposed an interpolation scheme based on radial basis functions, which can take pre-calculated adaptations and interpolate the changes so that another character can be processed quickly.

6.2 Limitations and Future Work

Even though the proposed approach has proven to be efficient for adapting motions, several aspects are still requiring hand interaction. In the future, we would like to improve the following points.

The foot plant detection algorithm can only handle motions for which at least one foot is planted at any time. This limitation obliged us to label by hand clips for which this assumption was not true. A robust labeling algorithm, working for any kind of motion would be most welcome.

The proposed motion adaptation algorithms may modify the height of the root joint of the character. In this case, the foot skating removal algorithm is not able to fix motion clip. Indeed, because it minimizes the height of the planted vertices it cannot handle a change applied on a few frames only. Instead of using existing approaches which also modify the motion of the limbs, we would like to extend the adaptation algorithm so that the planted feet remain in contact with the ground even after the adaptation.

The adaptation process requires that control vertices are placed along the motion. We automated this process by placing at least two control points per walk cycle. This has proven to be efficient, but the automatic placement fails in case the motion does not resemble enough a regular walk. Thus we would like to improve the placement algorithm, so that it is efficient on all motions.

The adaptation process does not take the strength of the character into account. For this, we should calculate the torques which is exerted by the muscles, which is something very difficult to do in a realistic fashion. This would allow the algorithm to detect whether a motion is doable by such or such character, instead of providing a solution even if the motion is impossible for the given target shape.

The current implementation of the motion adaptation algorithm makes it usable by computer scientists only as it requires the use of C++ programming. We would like to move this implementation to 3D Studio Max so that it becomes accessible to designers. This would also enable a user to repair a bad foot planting or a bad placement of the control points without having to recompile a program.

APPENDIX A

Publications

Following are the publications made during this thesis work:

- [KGE+04] H. Kim, T. Di-Giacomo, A. Egges, E. Lyard, S. Garchery, N. Magnenat-Thalmann. *Believable Virtual Environment: Sensory and Perceptual Believability.* Proc. CapTech Workshop on Modeling and Motion Capture Techniques for Virtual Environments, Believability in Virtual Environment, Zermatt, Switzerland, 2004.
- [MTLVL07] N. Magnenat-Thalmann, C. Luible, P. Volino, E. Lyard. *From Measured Fabric to the Simulation of Cloth.* 12th IEEE Inter. Conference on Emerging Technologies and Factory Automation, IEEE Publishing, 2007.
- [LMT07] E. Lyard, N. Magnenat-Thalmann. *A simple footskate removal method for virtual reality applications.* The Visual Computer, Springer-Verlag, Vol. 23, No. 9, pp. 689-695, 2007.
- [MTLKV08] N. Magnenat-Thalmann, E. Lyard, M. Kasap, P. Volino. *Adaptive Body, Motion and Cloth*. Proceedings of MIG08 workshop, 2008.
- [LMT08] E. Lyard, N. Magnenat-Thalmann. *Motion Adaptation Based on Character Shape*. Computer Animation and Virtual Worlds, Vol.19(3), pp. 189-198, John Wiley and Sons Ltd, 2008.

APPENDIX B

Derivation of the R-Formulae

The R-formulae state the following:

$$
\begin{aligned}
a\cos x + b\sin x &= R\cos(x-k)\\
a\cos x - b\sin x &= R\cos(x+k)\\
a\sin x + b\cos x &= R\sin(x+k)\\
a\sin x - b\cos x &= R\sin(x-k)
\end{aligned}
$$

with:

$$
\left\{
\begin{array}{rcl}
R &=& \sqrt{a^2+b^2}\\
k &=& \tan^{-1}(\frac{b}{a}
\end{array}
\right.
$$

This can be derived as follow:

$$
a\cos x + b\sin x = R\cos(x-k) = R\cos x\cos k + R\sin x\sin k
$$

hence:

$$
a = R\cos k \tag{B.1}
$$

$$
b = R\sin k \tag{B.2}
$$

$(B.1)^2 + (B.2)^2 \rightarrow$

$$
\begin{array}{rcl}
(R\cos k)^2 + (R\sin k)^2 &=& a^2+b^2\\
R^2\cos^2 k + R^2\sin^2 k &=& a^2+b^2\\
R^2(\cos^2 k + \sin^2 k) &=& a^2+b^2\\
R &=& \sqrt{a^2+b^2}
\end{array}
$$

$$\tan k = \frac{\sin k}{\cos k} = \frac{R \sin k}{R \cos k} = \frac{b}{a}$$

$$k = \tan^{-1} \frac{b}{a}$$

APPENDIX C

Calculation of the angular corrections

As depicted on figure 3.9, the rotations involved in the displacement of the foot feature the following vectors:

$$\begin{cases} V_1 &= P_1P_0 \\ V_2 &= P_2P_0 \\ V_3 &= P_3P_0 \end{cases} \tag{C.1}$$

Each extra rotation R_i that will be applied on a particular joint will move its associated vector V_i by an amount ΔV_i. Hence, the displacement of the end effector P_0 can be expressed in terms of ΔV_i as follow:

$$\Delta P_0 = \Delta V_1 + \Delta V_2 + \Delta V_3 \tag{C.2}$$

In the case of legs or arms, the three joints consituting the limb (hip, knee and ankle for the leg and shoulder, elbow and wrist for the arm) are hierarchically connected one after the other. This makes the rotations pile up while traversing the hierarchy and it allows to express the ΔV_i in terms of R_i and V_i:

$$\begin{cases} \Delta V_1 &= R_1R_2R_3V_1 - R_2R_3V_1 &= (R_1 - Id)R_2R_3V_1 \\ \Delta V_2 &= R_2R_3V_2 - R_3V_2 &= (R_2 - Id)R_3V_2 \\ \Delta V_3 &= R_3V_3 - V_3 &= (R_3 - Id)V_3 \end{cases} \tag{C.3}$$

As stated previously,The orientation of V_1 must be kept, hence the sum of all three rotations R_1, R_2 and R_3 must be the identity:

$$\begin{aligned} R_1R_2R_3 &= Id \\ R_2R_3 &= R_1^T \end{aligned} \tag{C.4}$$

Combining equations C.3 and C.4 gives:

$$\begin{cases} \Delta V_1 &= V_1 - R_1^TV_1 &= (Id - R_1^T)V_1 \\ \Delta V_2 &= R_1^TV_2 - R_3V_2 &= (R_1^T - R_3)V_2 \\ \Delta V_3 &= R_3V_3 - V_3 &= (R_3 - Id)V_3 \end{cases} \tag{C.5}$$

which enables one to express ΔP_0 using equations C.2 and C.5 as:

$$\begin{array}{rcl} \Delta P_0 & = & (R_3 - Id)V_3 + (R_1^T - R_3)V_2 + (Id - R_1^T)V_1 \\ & = & (0, d_y, d_z)^T \end{array} \tag{C.6}$$

Here the targeted displacement ΔP_0 is $(0, d_y, d_z)^T$ as the displacement takes place in the (Y, Z) plane and the rotation R_i is along the X axis. If one put $V_i = (0, y_i, z_i)^T$ and α_i the actual angle that R_i will rotate around the X axis then one can develop equation C.6 in 3D coordinates as follow:

$$\begin{array}{rcl} \Delta P_0 & = & \left(\left(\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha_3 & \sin\alpha_3 \\ 0 & -\sin\alpha_3 & \cos\alpha_3 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \right) \begin{pmatrix} 0 \\ y_3 \\ z_3 \end{pmatrix} \right. + \\ & & \left(\left(\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha_1 & -\sin\alpha_1 \\ 0 & \sin\alpha_1 & \cos\alpha_1 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha_3 & \sin\alpha_3 \\ 0 & -\sin\alpha_3 & \cos\alpha_3 \end{pmatrix} \right) \begin{pmatrix} 0 \\ y_2 \\ z_2 \end{pmatrix} \right. + \\ & & \left. \left(\left(\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha_1 & -\sin\alpha_1 \\ 0 & \sin\alpha_1 & \cos\alpha_1 \end{pmatrix} \right) \begin{pmatrix} 0 \\ y_1 \\ z_1 \end{pmatrix} \right. \end{array}$$

$$\begin{array}{rcl} \Delta P_0 & = & \begin{pmatrix} 0 & 0 & 0 \\ 0 & \cos\alpha_3 - 1 & \sin\alpha_3 \\ 0 & -\sin\alpha_3 & \cos\alpha_3 - 1 \end{pmatrix} \begin{pmatrix} 0 \\ y_3 \\ z_3 \end{pmatrix} + \\ & & \begin{pmatrix} 0 & 0 & 0 \\ 0 & \cos\alpha_1 - \cos\alpha_3 & -\sin\alpha_1 - \sin\alpha_3 \\ 0 & \sin\alpha_1 + \sin\alpha_3 & \cos\alpha_1 - \cos\alpha_3 \end{pmatrix} \begin{pmatrix} 0 \\ y_2 \\ z_2 \end{pmatrix} + \\ & & \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 - \cos\alpha_1 & \sin\alpha_1 \\ 0 & -\sin\alpha_1 & 1 - \cos\alpha_1 \end{pmatrix} \begin{pmatrix} 0 \\ y_1 \\ z_1 \end{pmatrix} \end{array}$$

$$\begin{array}{rcl} \Delta P_0 & = & \begin{pmatrix} 0 \\ (\cos\alpha_3 - 1)y_3 + \sin\alpha_3 z_3 \\ -\sin\alpha_3 y_3 + (\cos\alpha_3 - 1)z_3 \end{pmatrix} + \\ & & \begin{pmatrix} 0 \\ (\cos\alpha_1 - \cos\alpha_3)y_2 - (\sin\alpha_1 + \sin\alpha_3)z_2 \\ (\sin\alpha_1 + \sin\alpha_3)y_2 + (\cos\alpha_1 - \cos\alpha_3)z_2 \end{pmatrix} + \\ & & \begin{pmatrix} 0 \\ (1 - \cos\alpha_1)y_1 + \sin\alpha_1 z_1 \\ -\sin\alpha_1 y_1 + (1 - \cos\alpha_1)z_1 \end{pmatrix} \end{array}$$

$$\Delta P_0 = \begin{pmatrix} 0 \\ (\cos\alpha_3 - 1)y_3 + \sin\alpha_3 z_3 + (\cos\alpha_1 - \cos\alpha_3)y_2 - (\sin\alpha_1 + \sin\alpha_3)z_2 + (1 - \cos\alpha_1)y_1 + \sin\alpha_1 z_1 \\ -\sin\alpha_3 y_3 + (\cos\alpha_3 - 1)z_3 + (\sin\alpha_1 + \sin\alpha_3)y_2 + (\cos\alpha_1 - \cos\alpha_3)z_2 - \sin\alpha_1 y_1 + (1 - \cos\alpha_1)z_1 \end{pmatrix}$$

$$\Delta P_0 = \begin{pmatrix} 0 \\ \cos\alpha_3 y_3 - y_3 + \sin\alpha_3 z_3 + \cos\alpha_1 y_2 - \cos\alpha_3 y_2 - \sin\alpha_1 z_2 - \sin\alpha_3 z_2 + y_1 - \cos\alpha_1 y_1 + \sin\alpha_1 z_1 \\ -\sin\alpha_3 y_3 + \cos\alpha_3 z_3 - z_3 + \sin\alpha_1 y_2 + \sin\alpha_3 y_2 + \cos\alpha_1 z_2 - \cos\alpha_3 z_2 - \sin\alpha_1 y_1 + z_1 - \cos\alpha_1 z_1 \end{pmatrix}$$

Only the Y and Z components of ΔP_0 are kept and when put in relation with the targeted displacement $(0, d_y, d_z)^T$ this gives:

$$\begin{cases} \cos\alpha_3(y_3 - y_2) + \sin\alpha_3(z_3 - z_2) + \cos\alpha_1(y_2 - y_1) - \sin\alpha_1(z_2 - z_1) = y_3 - y_1 + d_y \\ \cos\alpha_3(z_3 - z_2) - \sin\alpha_3(y_3 - y_2) + \cos\alpha_1(z_2 - z_1) + \sin\alpha_1(y_2 - y_1) = z_3 - z_1 + d_z \end{cases} \quad \text{(C.7)}$$

If one puts the following:

$$\begin{cases} a_3 = z_3 - z_2 \\ b_3 = y_3 - y_2 \\ R_3 = \sqrt{a_3^2 + b_3^2} \\ k_3 = \tan^{-1}(\frac{b_3}{a_3}) \end{cases} \quad \text{and} \quad \begin{cases} a_1 = y_2 - y_1 \\ b_1 = z_2 - z_1 \\ R_1 = \sqrt{a_1^2 + b_1^2} \\ k_1 = \tan^{-1}(\frac{b_1}{a_1}) \end{cases}$$

it enables to rewrite equation C.7 as follow:

$$\begin{cases} b_3\cos\alpha_3 + a_3\sin\alpha_3 + a_1\cos\alpha_1 - b_1\sin\alpha_1 = y_3 - y_1 + d_y \\ a_3\cos\alpha_3 - b_3\sin\alpha_3 + b_1\cos\alpha_1 + a_1\sin\alpha_1 = z_3 - z_1 + d_z \end{cases}$$

If one further notices (see annexe B) that:

$$\begin{cases} a_3\sin\alpha_3 + b_3\cos\alpha_3 = R_3\sin(\alpha_3 + k_3) \\ a_3\cos\alpha_3 - b_3\sin\alpha_3 = R_3\cos(\alpha_3 + k_3) \\ a_1\cos\alpha_1 - b_1\sin\alpha_1 = R_1\cos(\alpha_1 + k_1) \\ a_1\sin\alpha_1 + b_1\cos\alpha_1 = R_1\sin(\alpha_1 + k_1) \end{cases}$$

Then equation C.7 becomes:

$$\begin{cases} R_3\sin(\alpha_3 + k_3) + R_1\cos(\alpha_1 + k_1) = y_3 - y_1 + d_y \\ R_3\cos(\alpha_3 + k_3) + R_1\sin(\alpha_1 + k_1) = z_3 - z_1 + d_z \end{cases}$$

which gives:

$$\sin(\alpha_3 + k_3) = \frac{y_3 - y_1 + d_y - R_1\cos(\alpha_1 + k_1)}{R_3} \quad \text{(C.8)}$$

$$\cos(\alpha_3 + k_3) = \frac{z_3 - z_1 + d_z - R_1\sin(\alpha_1 + k_1)}{R_3} \quad \text{(C.9)}$$

Moreover, if one notices that $\cos^2 x + \sin^2 x = 1$ and put $\begin{cases} y_3 - y_1 + d_y = y \\ z_3 - z_1 + d_z = z \end{cases}$, it yields to:

$$\left(\frac{y - R_1\cos(\alpha_1 + k_1)}{R_3}\right)^2 + \left(\frac{z - R_1\sin(\alpha_1 + k_1)}{R_3}\right)^2 = 1$$

$$\frac{y^2 + R_1^2(\cos^2(\alpha_1 + k_1) + \sin^2(\alpha_1 + k_1)) + z^2 - 2R_1(y\cos(\alpha_1 + k_1) + z\sin(\alpha_1 + k_1))}{R_3^2} = 1$$

If one put $y\cos(\alpha_1+k_1)+z\sin(\alpha_1+k_1) = R\sin(\alpha_1+k_1+k)$ with $\left\{ \begin{array}{rcl} R & = & \sqrt{y^2+z^2} \\ k & = & \tan^{-1}(\frac{y}{z}) \end{array} \right.$
Then:

$$\frac{y^2 + z^2 + R_1^2 - 2RR_1\sin(\alpha_1 + k_1 + k)}{R_3^2} = 1$$

$$\sin(\alpha_1 + k_1 + k) = \frac{R_3^2 - y^2 - R_1^2 - z^2}{-2RR_1} \tag{C.10}$$

And:

$$\alpha_1 = sin^{-1}\left(\frac{y^2 + z^2 + R_1^2 - R_3^2}{2RR_1}\right) - k_1 - k \tag{C.11}$$

with:

$$\left\{ \begin{array}{rcl} y & = & y_3 - y_1 + d_y \\ z & = & z_3 - z_1 + d_z \\ R_1 & = & \sqrt{(y_2 - y_1)^2 + (z_2 - z_1)^2} \\ R_3 & = & \sqrt{(z_3 - z_2)^2 + (y_3 - y_2)^2} \\ R & = & \sqrt{y^2 + z^2} \\ k & = & \tan^{-1}(\frac{y}{z}) \end{array} \right.$$

Now that α_1 is known, α_3 can be extracted from C.8 as:

$$\alpha_3 = \sin^{-1}\left(\frac{y - R_1\cos(\alpha_1 + k_1)}{R_3}\right) - k_3 \tag{C.12}$$

with:

$$\left\{ \begin{array}{rcl} y & = & y_3 - y_1 + d_y \\ k_1 & = & \tan^{-1}(\frac{b_1}{a_1}) \\ b_1 & = & z_2 - z_1 \\ a_1 & = & y_2 - y_1 \\ a_3 & = & z_3 - z_2 \\ b_3 & = & y_3 - y_2 \\ k_3 & = & \tan^{-1}(\frac{b_3}{a_3}) \end{array} \right.$$

Eventually:

$$\alpha_2 = -\alpha_1 - \alpha_3 \tag{C.13}$$

The above results (equations C.11, C.12, C.13) directly gives the angular correction that must be applied in 2D so that the end effector moves to the target point. Of course, because the computation was made considering only a rotation around the X axis and with the assumption that the segments lie on a 2D plane, then this precondition must be satisfied as much as possible if one wants to get satisfying results.

APPENDIX D

Skinning in Collada

Here is an example of skinning data stored in a Collada file:

```
<library_controllers>
  <controller id="Body-mesh-skin">
    <skin source="#Body-mesh">
      <bind_shape_matrix> 1  − 0 0 0 0 1 0 2.32921 ... < /bind_shape_matrix>
      <source id="Body-mesh-skin-joints">
        <Name_array id="Body-mesh-skin-joints-array" count="27">
          Bip01-node Bip01_Head-node Bip01_L_Calf-node Bip01_L_Clavicle-node ...
        < /Name_array>
        <technique_common>
          <accessor count="27" source="#Body-mesh-skin-joints-array">
              <param name="JOINT" type="Name"/ >
          < /accessor>
        < /technique_common>
      < /source>
      <source id="Body-mesh-skin-bind_poses">
        <float_array id="Body-mesh-skin-bind_poses-array" count="432">
          1e-006 -1 0 -1.41822 1 1e-006 0 1e-006 0 0 1 -86.5841  ...
        < /float_array>
        <technique_common>
          <accessor count="27" source="#Body-mesh-skin-bind_poses-array" stride="16">
              <param name="TRANSFORM" type="float4x4"/ >
          < /accessor>
        < /technique_common>
      < /source>
      <source id="Body-mesh-skin-weights">
        <float_array id="Body-mesh-skin-weights-array" count="8105">
          1 0.021049 0.97895 0.484387 0.515613 0.020732 ...
        < /float_array>
```

```
        <technique_common>
          <accessor count="8105" source="#Body-mesh-skin-weights-array">
              <param name="WEIGHT" type="float" />
          </accessor>
        </technique_common>
      </source>
      <joints>
        <input semantic="JOINT" source="#Body-mesh-skin-joints" />
        <input semantic="INV_BIND_MATRIX" source="#Body-mesh-skin-bind_poses" />
      </joints>
      <vertex_weights count="3801">
        <input offset="0" semantic="JOINT" source="#Body-mesh-skin-joints" />
        <input offset="1" semantic="WEIGHT" source="#Body-mesh-skin-weights" />
        <vcount>
          2 2 3 1 1 2 2 2 2 2 2 2 2 2 2 ...
        </vcount>
        <v>
          13 1 19 2 13 3 19 4 13 5 14 6 19 7 ...
        </v>
      </vertex_weights>
    </skin>
  </controller>
</library_controllers>
```

APPENDIX E

Prototypes of the `AnimOptimizer` functions

```
unsigned int loadAnimation(double sampling, osg::Node* root)
```
Returns the number of samples taken throughout the animation. `sampling` is the time interval between 2 samples.

```
void swapAnimation().
void removeFootSkating()

   void findFootSoleVertices()

   void calculatePlantedVertices()


void scaleCharacter(double scyeScale,
                    double upperArmScale,
                    double foreArmScale,
                    double waistKneeScale,
                    double insideLegScale,
                    double waistHeightScale,
                    double backLengthScale)


void scaleSkeleton(double scyeScale,
                   double upperArmScale,
                   double foreArmScale,
                   double waistKneeScale,
                   double insideLegScale,
                   double waistHeightScale,
                   double backLengthScale)


   void removeLeftArmPenetrations(char* filename)
```

```
void removeRightArmPenetrations(char* filename)

void removeLegsPenetrations(char* filename)

void optimizeBalance(char* filename)
```

the parameter `filename` of the 4 above functions is a name of the file from which the corrections should be loaded. In case the corrections must be recalculated from scratch, then `NULL` should be passed. After re-calculating the corrections, each of these function will save them in a file called *XXXXCorrectionsOutput.txt* with *XXXX* being either *leftArm*, *rightArm*, *legs* or *balance* depending on the function outputting the file.

```
void removeCollision(std::string penetratingJoint,
                     double penetratingJointRadius,
                     osg::Vec3d offset,
                     std::vector<std::string> jointsToCheck,
                     std::vector<double> jointsRadius,
                     std::vector<std::string> cylindersToCheck,
                     std::vector<double> cylindersRadius,
                     std::vector<std::string> jointsToModify,
                     std::vector<unsigned int> dof,
                     std::vector<double> weights,
                     std::vector<std::string> sameJoint,
                     std::vector<unsigned int> sameDof,
                     std::vector<bool> samePlus,
                     std::vector<unsigned int> sameJointIndex,
                     unsigned int sampling)
```

with `penetratingJoint` the name of the joint to monitor, `penetratingJointRadius` the minimal acceptable distance to keep, `offset` a vector for placing the center of the sphere that must remain empty, `jointsToCheck` the collection of joint that might collide, `jointsRadius` their size, `cylindersToCheck` the collection of cylinders that might collide (a cylinder is composed of two joints names), `cylindersRadius` the size of each cylinder, `jointsToModify` the joints that can be modified to remove the collisions, `dof` the degree of freedom that should be used (if a joint can be modified in more than one direction it should be duplicated here), `weights` the weight associated with each joint.
Several joints can be moved by the same amount as another joint. These should be given by `sameJoint`, the associated DoFs `sameDof`, `samePlus` whether the correction or its inverse should be applied and `sameJointIndex` the index of the joint to mimic. Eventually, `sampling` give the spacing between two control points.

```
void loadInterpolationData(unsigned int rightArmCPs),
                           unsigned int leftArmCPs,
                           unsigned int hipCPs,
                           unsigned int thighCPs,
                           unsigned int balanceCPs)
```

with the values passed as parameter calculated using the `calculateControlPoints()` method. Once the corrections are loaded, they can be interpolated with:

```
void interpolateCorrections(double rightArmScale),
                            double leftArmScale,
                            double rightLegScale,
                            double leftLegScale,
                            double trunkScale,
                            Interpolator interpol)
```

with the 5 first parameters being the scale applied to the character, and `interpol` the previously initialized interpolator.

APPENDIX F

Results of the performance evaluation per type of motion.

	Normal walk	360^o rotation	Run in circle	Lean
Head	0	0	0	0
Right Arm	0	9090	8363	13912
Left Arm	12864	8804	0	13446
Trunk	13419	27288	0	11270
Left Leg	37413	18568	26754	0
Right Leg	104	50	28682	24768

Table F.1: Mass distribution per motion per limb calculated through optimization, in grammes.

	Slow Walk	Fast walk	Jumps	Skip
Head	4922	0	3074	0
Right Arm	1866	0	0	16737
Left Arm	5162	0	2564	12845
Trunk	33127	58417	6777	0
Left Leg	8391	0	26615	34217
Right Leg	10323	5383	5383	0

Table F.2: Mass distribution per motion per limb calculated through optimization, grammes.

	Cylinders	Ellipsoids	Optimization
Normal walk	0.77 (0.34)	0.93 (0.44)	6.61 (3.38)
360^o rotation	2.38 (3.46)	3.18 (3.06)	7.81 (5.84)
Run in circle	1.77 (0.59)	0.97 (0.51)	17.25 (5.18)
Lean	1.02 (0.40)	1.33 (0.79)	9.20 (4.41)
Slow Walk	0.63 (0.31)	0.74 (0.38)	1.85 (0.28)
Fast walk	0.84 (0.42)	1.02 (0.42)	3.54 (2.02)
Jumps	2.21 (1.05)	2.77 (1.18)	13.54 (6.74)
Skip	1.73 (0.56)	1.80 (0.90)	12.89 (10.76)

Table F.3: Distances per motion from the real trajectory of the ZMP, in centimeters (SD).

	Distance	SD
Normal walk	2.33	2.33
360^o rotation	0.62	1.81
Run in circle	12.8	11.53
Lean	1.34	6.20
Slow Walk	0.01	0.17
Fast walk	2.62	4.86
Jumps	6.55	9.47
Skip	3.14	8.01)

Table F.4: Average distances from the supporting area of the real ZMP trajectory, in centimeters (SD).

Bibliography

[ALP06] Y. Abe, C. Karen Liu, and Z. Popović. Momentum-based parameterization of dynamic character motion. *Graph. Models*, 68(2):194–211, Academic Press Professional, Inc., 2006.

[AM00] M. Alexa and W. Müller. Representing animations by principal components. *Computer Graphics Forum*, 19(3):291–301, Eurographics Association, 2000.

[ACP03] B. Allen, B. Curless, and Z. Popović. The space of human body shapes: reconstruction and parameterization from range scans. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 587–594, New York, NY, USA, 2003. ACM Press.

[AF02] O. Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Trans. Graph.*, 21(3):483–490, ACM Press, 2002.

[AFO03] O. Arikan, D. A. Forsyth, and J. F. O'Brien. Motion synthesis from annotations. *ACM Trans. Graph.*, 22(3):402–408, ACM Press, 2003.

[tw08] Ascension technologies website. http://www.ascension-tech.com. accessed May 2008, 2008.

[BPW93] N. I. Badler, C. B. Phillips, and B. L. Webber. *Simulating humans: computer graphics animation and control*. Oxford University Press, Inc., New York, NY, USA, 1993.

[BB98a] P. Baerlocher and R. Boulic. Task-priority formulations for the kinematic control of highly redundant articulated structures. In *International Conference on Intelligent Robots and Systems*, pages 323–329. IEEE Press, 1998.

[BB04] P. Baerlocher and R. Boulic. An inverse kinematics architecture enforcing an arbitrary number of strict priority levels. *Vis. Comput.*, 20(6):402–417, Springer-Verlag New York, Inc., 2004.

[Bet01] J. T. Betts. *Practical methods for optimal control using nonlinear programming*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[BFS00] S. Biasotti, B. Falcidieno, and M. Spagnuolo. Extended reeb graphs for surface understanding and description. In *DGCI '00: Proceedings of the 9th International Conference on Discrete Geometry for Computer Imagery*, pages 185–197, London, UK, 2000. Springer-Verlag.

[BB98b] R. Bindiganavale and N. I. Badler. Motion abstraction and mapping with spatial constraints. In *CAPTECH '98: Proceedings of the International Workshop on Modelling and Motion Capture Techniques for Virtual Environments*, pages 70–82, London, UK, 1998. Springer-Verlag.

[BT59] R. B. Blackman and J. W. Tukey. *The Measurement of Power Spectra, From the Point of View of Communications Engineering*. Dover Publications, 1959.

[BLHB03] R. Boulic, B. Le Callennec, M. Herren, and H. Bay. Experimenting Prioritized IK for Motion Editing. In *Annual Conference of the European association for Computer Graphics*. EUROGRAPHICS, Eurographics Association, 2003.

[BCH+95] R. Boulic, T. K. Capin, Z. Huang, P. Kalra, B. Lintermann, N. Magnenat-Thalmann, L. Moccozet, T. Molet, I. S. Pandzic, K. Saar, A. Schnitt, J. Shen, and D. Thalmann. The humanoid environment for interactive animation of multiple deformable human character. In *Computer Graphics Forum (Proc. Eurographics '95)*, volume 14, pages 337–348. Blackwell Publishing, August 1995.

[BHT97] R. Boulic, Z. Huang, and D. Thalmann. A Comparison of Design Strategies for 3D Human Motions. In *Human Comfort and Security of Information Systems ; Advanced interface for the Information Society*, pages 19–21. Springer-Verlag, 1997.

[BMTT90] R. Boulic, N. Magnenat-Thalmann, and D. Thalmann. A global human walking model with real-time kinematic personification. *The Visual Computer*, 6(6):344–358, Springer-Verlag, 1990.

[BT92] R. Boulic and D. Thalmann. Combined direct and inverse kinematic control for articulated figure motion editing. *Computer Graphics Forum*, 11(4):189–202, Blackwell Publishers, 1992.

[BH00] M. Brand and A. Hertzmann. Style machines. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 183–192, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[BW95] A. Bruderlin and L. Williams. Motion signal processing. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 97–104, New York, NY, USA, 1995. ACM Press.

[Bro08] Browzwear. http://www.browzwear.com/. accessed May 2008, 2008.

[BA03] M. D. Buhmann and M. J. Ablowitz. *Radial Basis Functions: Theory and Implementations*. Cambridge University Press, 2003.

[BW71] N. Burtnyk and M. Wein. Computer-Generated Key Frame Animation. *Journal of the Society of Motion Picture and Television Engineers*, 80(3):149–153, Society of Motion Picture and Television Engineers, March 1971.

[pptfMA08] Calcium Builds Strong Bones post processing tool from Motino Analysis. http://www.motionanalysis.com/applications/animation/film/calcium.html. accessed May 2008, 2008.

[CSMT98] S. Carion, G. Sannier, and N. Magnenat-Thalmann. Virtual humans in cyberdance. In *Proc. of Computer Graphics International '98*, pages 142–153. IEEE Publisher, 1998.

[dat08] The Carnegie Mellon Mocap database. http://mocap.cs.cmu.edu/. accessed May 2008, 2008.

[CR74] E. Catmull and R. Rom. A class of local interpolating splines. *Computer Aided Geometric Design*, pages 317–326, 1974.

[CH07] J. Chai and J. K. Hodgins. Constraint-based motion optimization using a statistical dynamic model. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 8, New York, NY, USA, 2007. ACM Press.

[CCM+75] R. F. Chandler, C. E. Clauser, J. T. McConville, H. M. Reynolds, and J. W. Young. Investigation of inertial properties of the human body. Technical report, 1975.

[CK00] K. Choi and H. Ko. Online motion retargeting. *Journal of Visualization and Computer Animation*, 11(5):223–235, John Wiley & Sons, 2000.

[Coh92] M. F. Cohen. Interactive spacetime control for animation. *SIGGRAPH Comput. Graph.*, 26(2):293–302, ACM Press, 1992.

[com08] The Collada community. http://www.collada.org/. accessed May 2008, 2008.

[DAP08] M. Da Silva, Y. Abe, and J. Popovic. Simulation of human motion data using short-horizon model-predictive control. In *Eurographics 2008, To appear*. Eurographics Association, 2008.

[Dem55] W. T. Dempster. Space requirements of the seated operator. Technical report, 1955.

[DGG03] T. K. Dey, J. Giesen, and S. Goswami. Shape segmentation and matching with flow discretization. In F. Dehne et al., editor, *Proc. Workshop Alg. Data Structures*, page 25–36. Springer-Verlag, 2003.

[EMM04] A. Egges, T. Molet, and N. Magnenat-Thalmann. Personalised real-time idle motion synthesis. In *PG '04: Proceedings of the Computer Graphics and Applications, 12th Pacific Conference*, pages 121–130, Washington, DC, USA, 2004. IEEE Computer Society.

[EBMT00] L. Emering, R. Boulic, T. Molet, and D. Thalmann. Versatile tuning of humanoid agent activity. *Computer Graphics Forum*, 19(4):231–242, Blackwell Publishers, 2000.

[EW58] N. Eshkol and A. Wachmann. *Movement notation*. Weidenfeld and Nicolson, 1958.

[FP03] A. C. Fang and N. S. Pollard. Efficient synthesis of physically valid human motion. *ACM Trans. Graph.*, 22(3):417–426, ACM Press, 2003.

[Fea83] R. Featherstone. Position and velocity transformations between robot end-effector coordinates and joint angles. *Int. J. of Robotics Research*, 2(2):35–45, MIT Press, 1983.

[GM85] M. Girard and A. A. Maciejewski. Computational modeling for the computer animation of legged figures. *SIGGRAPH Comput. Graph.*, 19(3):263–270, ACM Press, 1985.

[GBT06] P. Glardon, R. Boulic, and D. Thalmann. Robust on-line adaptive footplant detection and enforcement for locomotion. *Vis. Comput.*, 22(3):194–209, Springer-Verlag New York, Inc., 2006.

[Gle97] M. Gleicher. Motion editing with spacetime constraints. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 139–ff., New York, NY, USA, 1997. ACM Press.

[Gle98] M. Gleicher. Retargeting motion to new characters. In *Proceedings of SIGGRAPH 1998*, Computer Graphics Proceedings, Annual Conference Series, pages 33–42, New York, 1998. ACM, ACM Press/ ACM SIGGRAPH.

[Gle01] M. Gleicher. Motion path editing. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 195–202, New York, NY, USA, 2001. ACM Press.

[GL98] M. Gleicher and P. Litwinowicz. Constraint-based motion adaptation. *The Journal of Visualization and Computer Animation*, 9(2):65–94, John Wiley & Sons, 1998.

[GSKJ03] M. Gleicher, H. Shin, L. Kovar, and A. Jepsen. Snap-together motion: assembling run-time animations. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 702–702, New York, NY, USA, 2003. ACM Press.

[gmcs08] The gypsy 4 motion capture system. http://www.metamotion.com/gypsy/gypsy-motion-capture-system.htm. accessed May 2008, 2008.

[HE64] J. Hanavan and P. Ernest. A mathematical model of the human body. Technical report, 1964.

[HKG06] R. Heck, L. Kovar, and M. Gleicher. Splicing upper-body actions with locomotion. *Computer Graphics Forum*, 25(3):459–466, Blackwell Publishing, September 2006.

[HCGM06] A. Heloir, N. Courty, S. Gibet, and F. Multon. Temporal alignment of communicative gesture sequences: Research articles. *Comput. Animat. Virtual Worlds*, 17(3-4):347–357, John Wiley & Sons Ltd., 2006.

[HSKK01] M. Hilaga, Y. Shinagawa, T. Kohmura, and T. L. Kunii. Topology matching for fully automatic similarity estimation of 3d shapes. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 203–212, New York, NY, USA, 2001. ACM Press.

[Hod98] J. K. Hodgins. Animating human motion. *Scientific American*, 278(3):64–69, Scientific American, Inc, 1998.

[HPP05] E. Hsu, K. Pulli, and J. Popović. Style translation for human motion. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 1082–1089, New York, NY, USA, 2005. ACM Press.

[Hud96] J. L. Hudson. Biomechanics of balance: paradigms and procedures. In *Proceedings of the XIIIth International Symposium on Biomechanics in Sports*, pages 286–289. Routledge, 1996.

[IAF06] L. Ikemoto, O. Arikan, and D. Forsyth. Knowing when to put your foot down. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 49–53, New York, NY, USA, 2006. ACM Press.

[IF04] L. Ikemoto and D. A. Forsyth. Enriching a motion collection by transplanting limbs. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 99–108, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[pptfVP08] IQ post processing tool from Vicon Peak. http://www.vicon.com/products/viconiq.html. accessed May 2008, 2008.

[JL00] K. Jeong and S. Lee. Motion adaptation with self-intersection avoidance. In *International workshop on human modeling and animation*, pages 77–85, 2000.

[KCvO07] L. Kavan, S. Collins, J. Žára, and C. O'Sullivan. Skinning with dual quaternions. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 39–46, New York, NY, USA, 2007. ACM Press.

[KGE+04] H. Kim, T. Di Giacomo, A. Egges, E. Lyard, S. Garchery, and N. Magnenat-Thalmann. Believable virtual environment: Sensory and perceptual believability. In *Proc. CapTech Workshop on Modeling and Motion Capture Techniques for Virtual Environments, Believability in Virtual Environment, Zermatt, Switzerland*, December 2004.

[KPS03] T. Kim, S. Park, and S. Shin. Rhythmic-motion synthesis based on motion-beat analysis. *ACM Trans. Graph.*, 22(3):392–401, ACM Press, 2003.

[KG03] L. Kovar and M. Gleicher. Flexible automatic motion blending with registration curves. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 214–224, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[KGP02] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. *ACM Trans. Graph.*, 21(3):473–482, ACM Press, 2002.

[KSG02] L. Kovar, J. Schreiner, and M. Gleicher. Footskate cleanup for motion capture editing. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 97–104, New York, NY, USA, 2002. ACM Press.

[JNK+02] J. J. Kuffner Jr., K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue. Self-collision detection and prevention for humanoid robots. In *International Conference on Robotics and Automation*, pages 2265–2270. IEEE, 2002.

[Lab66] R. Laban. *Choreutics*. MacDonald and Evans, 1966.

[LZK05] W. Lam, F. Zou, and T. Komura. Motion editing with data glove. In *Proceedings of the SIGCHI International conference on advances in computer entertainment technology*, pages 337–342, New York, 2005. ACM, ACM Press.

[Las87] J. Lasseter. Principles of traditional animation applied to 3d computer animation. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 35–44, New York, NY, USA, 1987. ACM Press.

[LZT] C. Lawrence, J. L. Zhou, and A. L. Tits. User's guide for cfsqp version 2.5: A c code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints.

[LB06a] B. Le Callennec and R. Boulic. Interactive motion deformation with prioritized constraints. *Graph. Models*, 68(2):175–193, Academic Press Professional, Inc., 2006.

[LB06b] B. Le Callennec and R. Boulic. Robust kinematic constraint detection for motion data. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 281–290, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[LCR+02] J. Lee, J. Chai, P. S. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive control of avatars animated with human motion data. *ACM Trans. Graph.*, 21(3):491–500, ACM Press, 2002.

[LS99] J. Lee and S. Shin. A hierarchical approach to interactive motion editing for human-like figures. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 39–48, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[LWS02] Y. Li, T. Wang, and H. Shum. Motion texture: a two-level statistical model for character motion synthesis. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 465–472, New York, NY, USA, 2002. ACM Press.

[LHP06] K. C. Liu, A. Hertzmann, and Z. Popović. Composition of complex optimal multi-character motions. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 215–222, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[LP02] K. C. Liu and Z. Popović. Synthesis of complex dynamic character motion from simple animations. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 408–416, New York, NY, USA, 2002. ACM Press.

[LMT07] E. Lyard and N. Magnenat-Thalmann. A simple footskate removal method for virtual reality applications. *The Visual Computer*, 23(9):689–695, Springer-Verlag, 2007.

[LMT08] E. Lyard and N. Magnenat-Thalmann. Motion adaptation based on character shape. *Computer Animation and Virtual Worlds*, 19(3):189–198, Wiley, 2008.

[MTK95] N. Magnenat-Thalmann and P. Kalra. The simulation of a virtual tv presentor. In *Proc. Pacific Graphics '95*, pages 9–21. World Scientific Press, 1995.

[MTKM01] N. Magnenat-Thalmann, P. Kalra, and L. Moccozet. *Virtual Humans*, pages 54–79. Routledge, N. Terashima and J. Tiffin (eds.), October 2001.

[MLT88] N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics interface '88*, pages 26–33, Toronto, Ont., Canada, Canada, 1988. Canadian Information Processing Society.

[MTLVL07] N. Magnenat-Thalmann, C. Luible, P. Volino, and E. Lyard. From measured fabric to the simulation of cloth. In *12th IEEE Inter. Conference on Emerging Technologies and Factory Automation*, 2007.

[MTLKV08] N. Magnenat-Thalmann, E. Lyard, M. Kasap, and P. Volino. Adaptive body, motion and cloth. In *Proceedings of MIG08 workshop, Amsterdam, The Netherlands*, 2008.

[MTSC04] N. Magnenat-Thalmann, H. Seo, and F. Cordier. Automatic modeling of virtual humans and body clothing. *J. Comput. Sci. Technol.*, 19(5):575–584, Institute of Computing Technology, 2004.

[MPS06] J. McCann, N. S. Pollard, and S. Srinivasa. Physics-based motion retiming. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 205–214, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[MM] M. Meredith and S. Maddock. Real-time inverse kinematics: the return of the jacobian.

[MM05] M. Meredith and S. Maddock. Adapting motion capture data using weighted real-time inverse kinematics. *Computer Entertainment*, 3(1):5–5, ACM Press, 2005.

[MHMTT97] L. Moccozet, Z. Huang, N. Magnenat-Thalmann, and D. Thalmann. Virtual actors interaction with 3d worlds. In *Proc. MultiMedia Modeling '97*, pages 307–322. World Scientific Press, 1997.

[MPS+03] M. Mortara, G. Patan, M. Spagnuolo, B. Falcidieno, and J. Rossignac. Blowing bubbles for multi-scale analysis and decomposition of triangle meshes. *Algorithmica*, 38(1):227–248, Springer-Verlag New York, Inc., 2003.

[cw08] Motion Analysis corp. website. http://www.motinoanalysis.com/. accessed May 2008, 2008.

[MHHR07] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. Position based dynamics. *J. Vis. Comun. Image Represent.*, 18(2):109–118, Academic Press, Inc., 2007.

[NF05] M. Neff and E. Fiume. Aer: aesthetic exploration and refinement for expressive character animation. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 161–170, New York, NY, USA, 2005. ACM Press.

[NF06] M. Neff and E. Fiume. Methods for exploring expressive stance. *Graph. Models*, 68(2):133–157, Academic Press Professional, Inc., 2006.

[Nic93] R. W. Nickalls. A new approach to solving the cubic: Cardan's solution revealed. *The Mathematical Gazette*, 77:354–359, Mathematical Association, 1993.

[web08b] The OpenSceneGraph website. http://www.openscenegraph.org/. accessed May 2008, 2008.

[Opt08] Optitex. http://www.optitex.com/. accessed May 2008, 2008.

[PSS02] S. Park, H. Shin, and S. Shin. On-line locomotion generation based on motion blending. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 105–111, New York, NY, USA, 2002. ACM Press.

[Pau82] R. P. Paul. *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, Cambridge, MA, USA, 1982.

[PBCM05] M. Peinado, R. Boulic, B. Le Callennec, and D. Meziat. Progressive Cartesian Inequality Constraints for the Inverse Kinematics Control of Articulated Chains. In *Eurographics, Short Presentation session*, pages 93–96. Eurographics Association, 2005.

[PMM+07] M. Peinado, D. Meziat, D. Maupu, D. Raunhardt, D. Thalmann, and R. Boulic. Accurate on-line avatar control with collision anticipation. In *VRST '07: Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, pages 89–97, New York, NY, USA, 2007. ACM Press.

[PMRB06] M. Peinado, D. Meziat, D. Raunhardt, and R. Boulic. Environment-Aware Postural Control of Virtual Humans for Real-time Applications. In *of the SAE Conference on Digital Human Modeling for Design and Engineering*, 2006.

[web08a] Polhemus website. http://www.polhemus.com/. accessed May 2008, 2008.

[PPM+03] 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 *Computer Graphics International*, pages 96–104. IEEE Computer Society, 2003.

[Pop00] Z. Popovic. Controlling physics in realistic character animation. *Communications of the ACM*, 43(7):50–58, ACM Press, 2000.

[PW99] Z. Popović and A. Witkin. Physically based motion transformation. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 11–20, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[PFTV92] W. H. Press, B. P. Flannery, S. A. Tukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing 2nd ed.* Cambridge University Press, 1992.

[PB02] K. Pullen and C. Bregler. Motion capture assisted animation: texturing and synthesis. *ACM Trans. Graph.*, 21(3):501–508, ACM Press, 2002.

[RCB98] C. Rose, M. F. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Comput. Graph. Appl.*, 18(5):32–40, IEEE Computer Society Press, 1998.

[RL87] P. J. Rousseeuw and A. M. Leroy. *Robust regression and outlier detection*. John Wiley & Sons, Inc., New York, NY, USA, 1987.

[SH07] A. Safonova and J. K. Hodgins. Construction and optimal search of interpolated motion graphs. *ACM Trans. Graph.*, 26(3):106, ACM Press, 2007.

[SHP04] A. Safonova, J. K. Hodgins, and N. S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Trans. Graph.*, 23(3):514–521, ACM Press, 2004.

[SG92] T. W. Sederberg and E. Greenwood. A physically based approach to 2–d shape blending. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 25–34, New York, NY, USA, 1992. ACM Press.

[SCPMT01] H. Seo, F. Cordier, L. Philippon, and N. Magnenat-Thalmann. *Interactive Modelling of MPEG-4 Deformable Human Body Models*, pages 120–131. Kluwer Academic Publishers, 2001.

[SMT03] H. Seo and N. Magnenat-Thalmann. An automatic modeling of human bodies from sizing parameters. In *I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 19–26, New York, NY, USA, 2003. ACM Press.

[SLY05] C. H. Séquin, K. Lee, and J. Yen. Fair, g - and c -continuous circle splines for the interpolation of sparse data points. *Computer-Aided Design*, 37(2):201–211, Elsevier Science Ltd, 2005.

[SLW+06] J. A. Shepherd, Y. Lu, K. Wilson, T. Fuerst, H. Genant, T. N. Hangartner, C. Wilson, D. Hans, and E. S. Leib. Cross-calibration and minimum precision standards for dual-energy x-ray absorptiometry: the 2005 iscd official positions. *Journal of Clinical Densitometry*, 9:31–6, Springer-Verlag, 2006.

[SKG03] H. Shin, L. Kovar, and M. Gleicher. Physical touch-up of human motions. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 194, Washington, DC, USA, 2003. IEEE Computer Society.

[SL06] H. Shin and J. Lee. Motion synthesis and editing in low-dimensional spaces: Research articles. *Comput. Animat. Virtual Worlds*, 17(3-4):219–227, John Wiley & Sons Ltd., 2006.

[SLSG01] H. Shin, J. Lee, S. Shin, and M. Gleicher. computer puppetry: an importance-based approach. *ACM Transactions on Graphics*, 20(2):67–94, ACM Press, 2001.

[SKK91] Y. Shinagawa, T. L. Kunii, and Y. L. Kergosien. Surface coding based on morse theory. *IEEE Comput. Graph. Appl.*, 11(5):66–78, IEEE Computer Society Press, 1991.

[Sim06] D. Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley-Interscience, 2006.

[SRC01] P. J. Sloan, C. F. Rose III, and M. F. Cohen. Shape by example. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 135–143, New York, NY, USA, 2001. ACM Press.

[SKL07] K. Sok, M. Kim, and J. Lee. Simulating biped behaviors from human motion data. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 107, New York, NY, USA, 2007. ACM Press.

[Stu98] D. J. Sturman. Computer puppetry. *IEEE Computer Graphics and Applications*, 18(1):38–45, IEEE Computer Society Press, 1998.

[SM01] H. C. Sun and D. N. Metaxas. Automating gait generation. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 261–270, New York, NY, USA, 2001. ACM Press.

[TK05] S. Tak and H. Ko. A physically-based motion retargeting filter. *ACM Trans. Graph.*, 24(1):98–117, ACM Press, 2005.

[TSK00] S. Tak, O. Song, and H. Ko. Motion balance filtering. *Computer Graphics Forum*, 19(3):437–446(10), Blackwell Publishers, 2000.

[TGB00] D. Tolani, A. Goswami, and N. I. Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graph. Models Image Process.*, 62(5):353–388, Academic Press, Inc., 2000.

[vd96] P. van Overschee and B. L. de Moor. *Subspace identification for linear systems: theory, implementation, applications*. Kluwer Academic Publishers Group, Dordrecht, The Netherlands, 1996.

[msw08] Vicon Peak motion systems website. http://www.vicon.com/. accessed May 2008, 2008.

[Too08] VR Juggler-Open Source Virtual Reality Tools. http://www.vrjuggler.org/. accessed May 2008, 2008.

[VB04] M. Vukobratovic and B. Borovac. Zero-moment point - thirty five years of its life. *International Journal of Humanoid Robotics*, 1(1):157–173, World Scientific Publishing, 2004.

[WB03] J. Wang and B. Bodenheimer. An evaluation of a cost metric for selecting transitions between motion segments. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 232–238, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[WK88] A. Witkin and M. Kass. Spacetime constraints. In *Proceedings of SIGGRAPH 1988*, Computer Graphics Proceedings, Annual Conference Series, pages 159–168, New York, 1988. ACM, ACM Press/ ACM SIGGRAPH.

[WP95] A. Witkin and Z. Popovic. Motion warping. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 105–108, New York, NY, USA, 1995. ACM Press.

[YKH04] K. Yamane, J. J. Kuffner, and J. K. Hodgins. Synthesizing animations of human manipulation tasks. *ACM Trans. Graph.*, 23(3):532–539, ACM Press, 2004.

[YP03] K. Yin and D. K. Pai. Footsee: an interactive animation system. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 329–338, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[ZB94] X. Zhao and N. Badler. Interactive body awareness. *Computer-Aided Design*, 26(12):861–866, Elsevier Science Ltd, 1994.

[ZH03] V. B. Zordan and N. C. Van Der Horst. Mapping optical motion capture data to skeletal motion using a physical model. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 245–250, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.