Animating Character Locomotion Using Biomechanics-Based Figure Models

Alexander Savenko

MSc, BA (Hons) Moscow State University

A Thesis Submitted in Partial Fulfilment of the Requirement of the Award of PhD

July 2002

Abstract

Modelling human locomotion is not a new task in Computer Animation. During the last twenty years many methods aimed at solving this task have been elaborated. The evolution of these methods has mainly been directed at the use of more complicated mathematical apparatus, while little research has been done that questions some of the basic assumptions used in Character Animation, or performs a deeper investigation of the basis of the subject modelled. This thesis tries to correct this omission. Thus, the main aim of the project was not to introduce new computational algorithms, but to find out how the existing animation algorithms and models can benefit from a deeper understanding of human biomechanics.

The main innovations of this project lie in the area of character modelling and animation of human locomotion using Motion Capture (MC) data. In the area of modelling, the project revises the simplified robotics-based approach to figure modelling and identifies many inaccuracies in the robotics model. A new biomechanics-based figure model was then proposed and evaluated in the context of the MC-based animation.

The main results, however, were achieved in the area of MC-based animation. A new approach to motion retargetting was proposed, which suggested using biomechanics knowledge of human locomotion to improve the realism and universality of the method. It was also suggested how to utilise motion analysis techniques to extract the biomechanical information required for retargetting from the source motion data. The implementation of this approach achieved the desired goals, producing realistic animations of human locomotion.

Declaration

This thesis has been composed by the undersigned. It has not been accepted in any previous application for a degree. The work is original and all sources of information have been acknowledged.

Alexander Savenko

Contents

1 Introduction

1.1 Animation of Human Locomotion

Animation of human locomotion is one of the most important areas in Computer Graphics. The wide use of digital characters in various applications, such as Virtual Reality or the film industry, contributes to the continuously growing interest to this area. At the same time, the animation of human locomotion is an extremely challenging task. If we are inclined not to be very critical of the "realism" of unfamiliar or artificial motions (for instance, motions of dinosaurs, robots or cartoon characters), we would immediately notice any artefacts in an animation of such familiar activity as human locomotion. The problem of achieving high realism seems even more difficult, if the tremendous variety of human locomotion is considered.

Though animation of human locomotion is not a new task in Computer Graphics – it first appeared about 30 years ago – the search for a practical method to solve this task has not finished [Computer Graphics World, 2002]. Moreover, most modern techniques are still dealing with only the most basic cases of human locomotion and, even here, difficulties are encountered. However, one should not think that there is no progress in this area. On the contrary, the progress made during last twenty years is really impressive: starting as a small research area, which did not have any practical applications, it has become one of the most studied and widely-used areas of Computer Animation. The following review demonstrates the evolution of the technologies used in the animation of human locomotion and suggests the directions of future research.

The use of computers to represent the animation of human locomotion started at the end of the 1970s [Zeltzer, 1982]. Though all the main streams emerged approximately at the same time, they reached the peaks of their popularity at different times, so one can speak about three periods in the history of the area: kinematics, dynamics and motion-capture periods.

The 1980s passed under the badge of kinematics-based methods. The idea of the first generation of animation techniques was to generate a series of keyframes from biomechanics knowledge about the gait. So, the final animation was produced using a keyframe approach, i.e. using interpolation. The generation of keyframes (or key-postures) was performed using finite state machines that were controlled by high-level parameters, such as step length and cadence. This method was first proposed by Zeltzer in 1982, who implemented it in his Skeleton Animation system [Zeltzer, 1982]. Though it was simple, it produced quite good

animations. During the next few years, several improvements to this type of method were proposed; these were mainly related to improving the finite state machines, using skeleton models with more degrees of freedom, etc.

The next serious step in the area was to start using Inverse Kinematics (IK) algorithms. By that time, these were widely exploited in robotics, and in 1985, Girard and Maciejewski adopted them in Character Animation [Girard et al., 1985]. IK rapidly became one of the most prevailing tools in the animation of figure locomotion. It proved to be very effective for interactive manipulations with articulated figures. Furthermore, it was directly applied to the animation of figure locomotion: the IK algorithm (and later the Inverse Dynamics algorithm) was at the heart of a new approach to the animation of locomotion — goal-directed animation. The idea of this approach is to generate the motion from specified goals, rather than from explicitly- defined key-postures. An example of this technique is Girard's footstep-driven animation, which used IK to generate the transition of the character's limbs between procedurally-generated (but editable) footsteps[1]. [Character Studio™ R2. User's Guide., 1998]. The obvious benefit of goal-driven animation is that the control of the animation task is transferred to a higher level, i.e. from manipulation of a series of joint angles to manipulation with goals and constraints.

Unfortunately, pure IK-based methods produce rather unrealistic animation, which limits their use as a primary animation technique. However, in the 1990s another application of IK became very popular: here IK was used as a supplementary correction tool to post-process animations produced using Motion Capture [Gleicher, 1998].

Since kinematics methods did not meet the animators' requirements, other approaches started to attract more researchers. Thus, the end of 1980s and the first part of 1990s can be characterised as a period of investigating dynamics-based methods. The main advantage of these is that a dynamics algorithm generates motion in accordance with physical laws, which asserts realism of the resulting animation. Also, another advantage is that constraints can be naturally incorporated in dynamics methods.

Forward Dynamics algorithms need to know the forces and torques that are operating in order to perform the simulation. Since the muscle and ground reaction forces which produce locomotion are usually unknown, the only practical dynamics methods are those that can calculate these forces automatically, i.e. Inverse Dynamics based methods. In 1985, Isaacs

---

[1] This algorithm was later implemented in a commercial animation package, 3D Studio MAX.

and Cohen proposed the first inverse-dynamics system for the animation of articulated figures. This was followed by several works, in which dynamics were applied to animation of human walking and running [Bruderlin, 1989; Hodgins, 1995; Ko, 1996].

Another approach of using dynamics is to use it for the post-processing of animation to check the physical validity and ergonomic characteristics of motion generated by a kinematics algorithm. Though the quality of the animation is still mainly defined by a kinematic engine, the use of dynamics allows generalisation. For instance, walking animation can be adapted for walking on uneven ground or inclined slopes, walking with turns, walking with a load, etc.

In general, the following conclusion can be drawn from an analysis of different dynamics methods: these methods can create physically-correct animations, handle interaction with the environment, etc, but there are no guarantees that the animation produced will look realistic. Some causes of this are the many simplifications made in dynamics algorithms and the absence of data. However, since these can hardly be avoided at the current stage of technology, it is more practical to use dynamics as a supplementary tool rather than the primary simulation method.

When the majority of the research community became disappointed with the use of dynamics methods for animation of figure locomotion, Motion Capture (MC) methods started to dominate in the field. By that time, MC technology achieved very impressive results: MC systems became very accurate and easy to use. Also, the systems themselves had become more accessible for animators. Of course, these accelerated the developments in this area. Now, almost any animation system supports MC, and MC-controlled characters have populated many films, TV shows and video games.

The idea of MC is to capture a motion performed by a real actor, digitise it and apply it to a computer character. The fact that the motion used for the character is exactly the same as the original one is both the strongest and the weakest point of the approach. On the one hand it guarantees that the animation will be realistic, but on the other hand, it limits the number of possible applications since many of them need the motion to be modified or corrected, for instance, to adapt it for the virtual environment. Recent developments in the area are directed at overcoming this drawback, and several motion editing methods have been introduced. The general idea of these is to adapt a motion for the animator's needs while retaining the characteristics of the original motion.

Thus, by now, several approaches to the animation of figure locomotion have been formed. There are simulation approaches, which can be used to create almost arbitrary animations, but

whose quality is relatively low. There are MC-based methods, which are limited by the available motion data, but which can produce very realistic animations. And, finally, there are mixed approaches that combine all of these methods. In view of the drawbacks of the first two approaches, the last one looks the most promising. However, it is unlikely that notable results can be achieved here if the developments apply only to the mathematical and physical apparatus of the algorithms: human locomotion is a very complicated and multiform activity and, without doubt, any algorithm attempting to reproduce it should posses good knowledge about its basis.

1.2 Project Aims

The main aim of this project was to learn how knowledge of human anatomy and biomechanics could be used to create more realistic character animations and make the process of creating these animations more efficient and universal.

This investigation had three main parts. The task of the first part was to study the biomechanical basis of human motion, particularly joint motion, identify characteristics that could be important for animation and apply this knowledge to design a biomechanics-based figure model. This part of the project also involved implementation of the above model as a plug-in for 3D Studio MAX, a popular 3D-animation program.

The objective of the second part was to develop and implement a new method for animating human locomotion that would be based on knowledge of gait biomechanics and could take advantage of the novel features of the above figure model.

The task of the concluding part of this research was to combine the results of the modelling and animation parts and to evaluate the figure model from the point of view of animation quality and usability in the animation environment.

1.3 Contribution of the Project

This research has touched on many different topics, including Biomechanics, 3D Modelling, Inverse Kinematics, etc. However, the main results were achieved in the areas central for this project, Character Modelling and Animation using Motion Capture data.

A deeper investigation of human biomechanics has allowed the identification of many significant characteristics of human motion that had been ignored by researchers previously. All of these features have been incorporated in the figure model and evaluated. In this way, the figure model supports various anatomic types of joints, allows modelling such

particularities of joint motion as combined motion, simulates kinematic dependencies between different joints, etc. Though, not all of these innovations have proved to have a significant effect on the animation produced, the project has demonstrated how such features can be efficiently modelled, and suggested some areas of Character Animation where the effect of these features can be more important. Also, the work on the figure model has produced many useful practical innovations in the area of Character Modelling.

The main outcome of the animation part of the project was the development of a novel approach for adapting MC data to different figure models. The key idea of this approach is to use knowledge of human gait to enhance the captured data by reconstructing the structure of the original motion (gait phases, etc) and extracting information about the constraints, so the developed retargetting algorithm can accurately adjust the motion to the new model and environment. The proposed approach has proved to be very efficient for retargetting walking motions, producing realistic-looking animations, and removing many of the artifacts that may occur in the animations created from MC data

1.4 Organisation of the Thesis

The thesis is organised as follows. Chapter 2 provides a detailed overview of the problem area. It presents the main animation techniques and models that are used in Character Animation and analyses their advantages and disadvantages.

Chapter 3 provides an insight into human anatomy and the biomechanics of human motion. It concentrates on joint motion, providing information about the main human joints and analysing their role in gait. This knowledge serves as foundation for the creation of biomechanics-based figure models and algorithms.

Chapter 4 introduces the practical part of the project. First, it discusses the necessity of having a system for experimenting with models and algorithms created within this research, and formulates the main requirements for the system. This is followed by a short overview of 3D Studio MAX, which served as an environment for the system, and its SDK (Software Development Kit). The rest of the chapter considers the architecture of the system and describes the biomechanics-based figure model, the foundation of this system.

Chapters 5 and 6 go through all phases of animating the figure model. Chapter 5 starts with an overview of the Motion Capture techniques, focusing on the parts that raised the greatest problems during the development. Then, it introduces the retargetting problem and suggests that its solution requires knowing the structure of motion data and an understanding of the basis of the re-created motion. The rest of the chapter explains how the required information

can be extracted from the data, using motion analysis techniques. The novel retargetting algorithm that uses the biomechanical information extracted from the motion data is described in Chapter 6.

The evaluation of the presented figure model and the animation algorithm is provided in Chapter 7. It reviews the main innovations proposed in this research, such as support of the combined motion or use of motion analysis to derive constraints. The summary and the consideration of possible future work round off the chapter and the thesis.

2 Approaches in Character Animation

2.1 Model of a Character

Before we start describing different animation techniques, we must specify an object to animate. In principle, it should be a very realistic model of a human body with accurately modelled deformable skin and muscles, flowing hair, etc [Aubel, 2000; Magnenat-Thalmann et al., 2001; Hadap, 2001]. However, if we used such a model as a primary model for the animation of locomotion, we would have to deal with many additional tasks, not related to our main goal. That is why the area of Character Animation is subdivided to many sub-areas that deal with their particular tasks and work with their own separate models. This subdivision allows removing unnecessary details and to work with simplified models. As we are interested in the animation of locomotion our model will not contain anything that does not relate to modelling of locomotion.

The most popular model for figure animation is a skeleton model, that is, a mechanical representation of a human skeleton in the form of rigid links connected by joints. This model can be animated by changing the relative positions of the links and by animating the skeleton model as a solid object. In the same way as a real skeleton defines the general shape of the body, a skeleton model defines the shape (geometry) of the graphics model. Consequently, the skeleton animation can be mapped to the full-model animation: either the character model is modified in accordance with the skeleton transformations or it is procedurally recreated for every new position of the skeleton. The full process of character animation using a skeleton model looks like this:

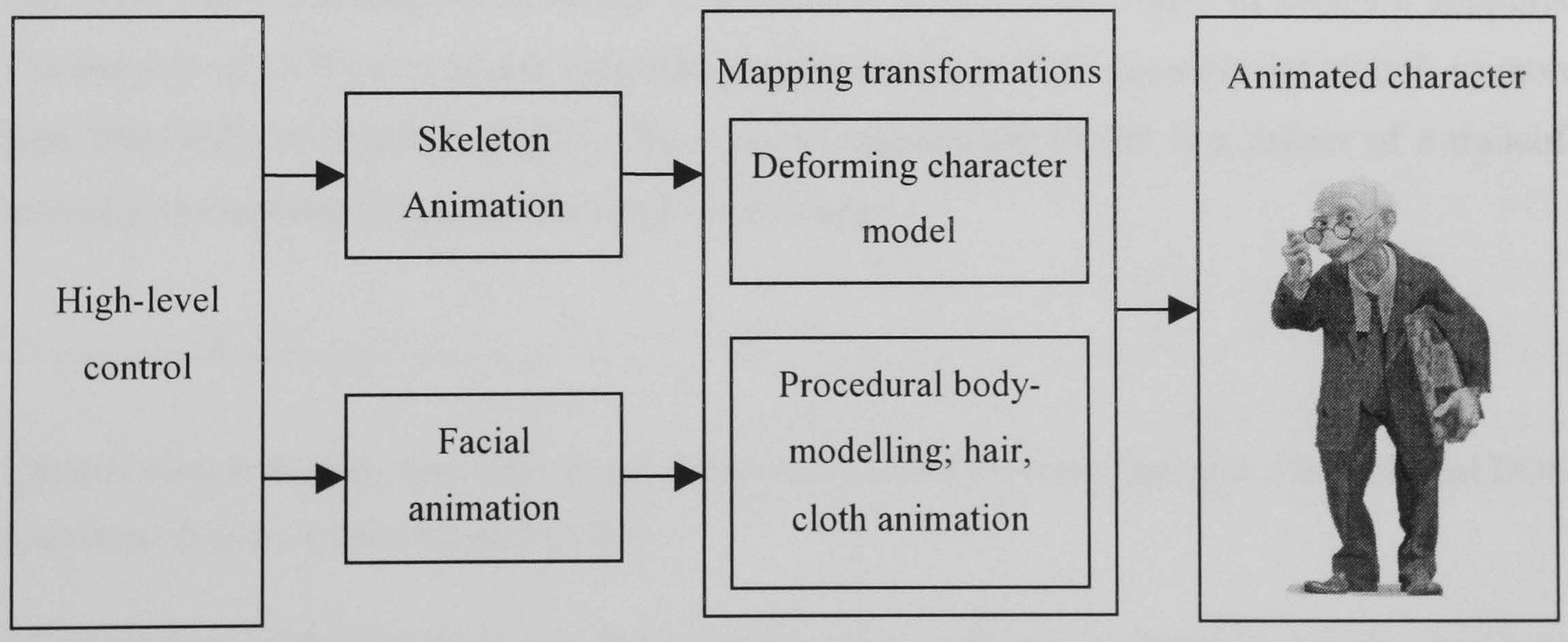


Figure 2.1: Character animation diagram

2.1.1 Skeleton Model

Although skeleton models can differ from one application to another, all of them are based on the same idea: a skeleton model is a coherent graph of joints that connects skeletons links (bones).

Each joint has up to 3 degrees[1] of freedom (DOFs), which allow links to rotate relative to each other. The root of the hierarchy (pelvis) has three additional translational DOFs, providing translational movement of the figure as a whole. In this way, a set of all the DOFs – called a state vector – completely defines the posture of the figure and its position and orientation in 3D-space.

The fact that graphs of joints have a tree structure[2] is a very important feature of the skeleton model. This kind of model structure is convenient for the calculation of transformations from a world reference frame to the reference frame of a joint. To find the matrix of this transformation we simply multiply transformation matrices of all joints between the root and the current joint.

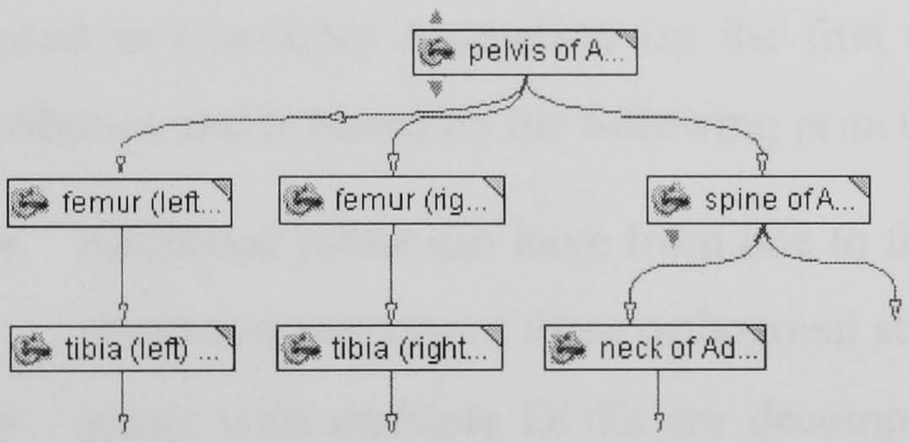


Figure 2.2: Tree structure of a skeleton model

Calculating joint-to-world transformations:

Matrix of foot-to-world frame transformation:
$^{w}T_{ankle} = T_{ankle} \cdot T_{knee} \cdot T_{hip} \cdot {}^{w}T_{pelvis}$

where T_x is a matrix of a local transformation from a reference frame of joint X to the reference frame of its parent, and $^{w}T_x$ transforms from the X-frame to the world frame.

The main factor defining the accuracy of a skeleton model is the range of DOFs it supports. The number of DOFs in a model varies from approximately 18 in the simplest models to more then 100 DOFs in detailed models. Choosing an appropriate model is a matter of a tradeoff between its accuracy (functionality) and its efficiency.

---

[1] Models used in medical applications may support all 6 DOFs for every joint (i.e. 3 translational DOFs in addition to three usual rotational DOFs)

[2] A real human skeleton is not a tree-like structure; some joints are combined in complex structures, which motions are interdependent on each other. Examples of these complexes are the lower arm, the shoulder complex and the foot complex. Though their real complexity is usually ignored in CA, there have been some works that address this problem [Boulic et al., 1998; Maurel, 1998].

The requirements of the application define which DOF has to be put in the model. For instance, the model of an avatar does not have to be very accurate, though it should have enough DOFs to provide a range of supported movements from simple walking to opening the mouth. In contrast, a model used in an ergonomic application has to be very accurate (for example, simulating every vertebra in the spine or every phalanx in the finger) but it may be limited to a part of the body.

The more complicated model is, the more difficult it is to deal with. The efficiency of simulation methods and tools (Inverse Kinematics and Dynamics) are particularly heavily dependent on complexity of the model: if a model has too many DOFs the algorithms may not be able to perform the calculations in reasonable time. Yet the main difficulty in using more complex models arises from the fact that different models have different problems and pitfalls; so adding just one joint or DOF to the model may require a considerable review of the algorithms.

The crucial part of the skeleton model concerns how transformations in joints are performed. The approach to applying transformations has hardly changed since the skeleton model was used in Character Animation for the first time. This mechanical model was adopted from robotics and is based on the following principles:

- rotational joints can have from one to three DOFs, allowing rotations about a maximum of three orthogonal static axes;
- joints with multiple DOFs are decomposed into several single-DOF joints;
- axes of joints are parallel to one of main figure planes; if one wants to work with a coupled motion, which takes place in two or three anatomical planes (Figure 2.3), one has to deal with three orthogonal DOFs rather than one.

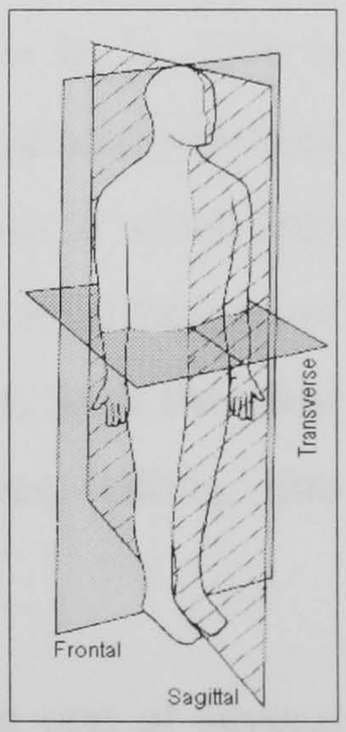


Figure 2.3: Anatomical planes

Though, on the face of it, these principles look quite acceptable, they are not anatomically accurate: real axes are not orthogonal to each other; they can even change during the motion; motions in joints with multiple DOFs can be interdependent (so modelling such DOFs as separate joints is not accurate). In spite of this, the robotics approach is generally used and only in a few works have there been attempts to use more accurate models [Maurel, 1998].

2.1.2 Realistic Body Model: Skin Generation, Muscle Modelling

After a skeleton model has been animated, it is necessary to apply the same motion to the (more realistic) body model that will be used for visualisation. There are two main approaches to creating realistic models of the human body. The traditional approach is to specify the body explicitly; the corresponding models can be surface models (polygon meshes, spline surfaces) or constructive solid geometry (CSG) models. The other approach is to use procedure-defined (or implicit) models [Sheepers et al., 1997; Fua et al., 1998; Nedel et al., 1998].

The obvious benefit of the procedural-model approach is that the model is defined for any posture rather than for one specific posture. In addition, since these models are based on anatomical principles, it is guaranteed that the model will look realistic for virtually any posture.

A procedural model consists of several layers. These layers are built around each other starting with a skeleton layer. Since the skeleton is not usually visible it does not have to have a graphical representation but if it does it will normally be portrayed by a predefined mesh or CSG model. The main purpose of the skeleton is to serve as a foundation for the tissue layer. As the name suggests, the body tissues (muscles, tendons, fat) are modelled on this layer. This is usually done by means of CSG objects (metaballs) that are stretched / squeezed in accordance with movements of the bones to which they are attached (Figure 2.4). The rules that control the deformations of the metaballs vary according to the implementation, with the most advanced models supporting several muscle types, taking into account tendons, etc [Scheepers et al., 1997].

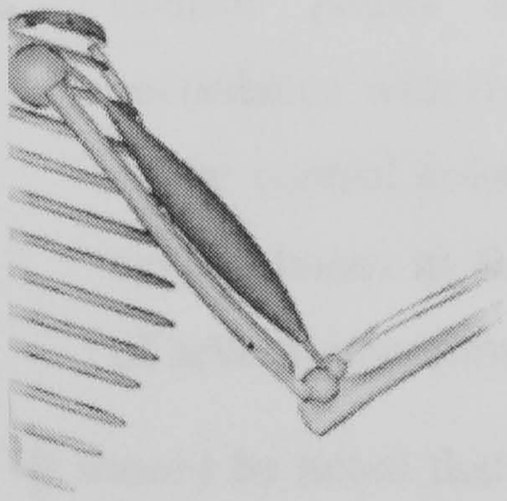


Figure 2.4: Muscle model

The dynamic skin layer completes the model. This layer is generated using one of several implicit-surface-generation algorithms. These produce a smooth surface that covers all the internal objects (bones, muscles, etc). Unfortunately the skin generated in this way will have artefacts and will not necessarily deform smoothly. Therefore, special algorithms are used to correct this basic surface to create a realistic skin [Wilhelms et al., 1997].

This process can be continued further to simulate hair, clothes, etc, so at the end of the modelling, a highly realistic anatomically-correct character model is produced. The fact that the model is created automatically is both an advantage and a drawback of the method. On the one hand, the process is automatic so the animator does not have to spend time doing modelling, but, on the other hand, this approach does not give the animator flexibility to use a specific model he/she may want.

In contrast to the procedural-model approach, the method of explicitly defined models is a general-purpose method and it has proved to be very powerful and flexible, allowing the creation of top-quality animations. In this method, a character model is manually created for a particular posture of the character and it is modified each time the posture changes. It would be an immense undertaking to modify a geometrical model manually for each posture of the figure. Fortunately there are 'deformable skin' algorithms that can do this job for us. These algorithms work as follows:

- Create a deformable geometrical model (skin) of a character. ('Deformable' means that the geometry should have control points whose transformations modify the geometry. For example, both Non-Uniform Rational B-Splines (NURBS) objects and triangle meshes are deformable).
- Create a skeleton model that matches the proportions and the posture of the geometrical model.
- Define how a particular bone (link) affects the surrounding skin (Figure 2.5).
- When a bone moves the algorithm transforms the control points of the geometrical model in accordance with its movement. If the transformation of the control point is affected by the movements of several bones its final transformation will be a blend of several transformations.

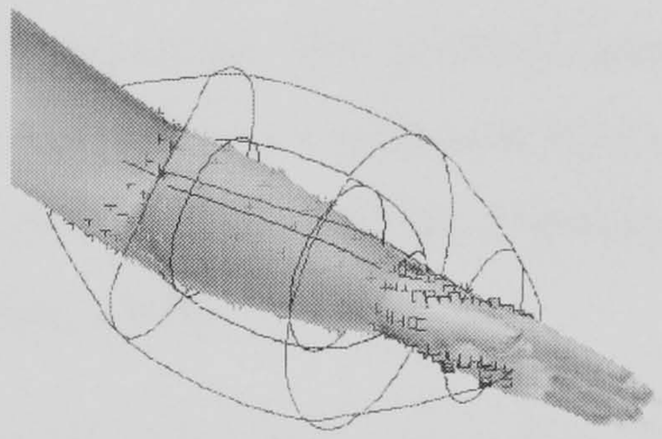


Figure 2.5: Attaching mesh to the skeleton model in 3D Studio Max™

It should be noted that the process of attaching a skin to the skeleton model is not easy and requires a lot of manual tuning, but sometimes even the finest tuning cannot make the deformed model look very realistic. The reason is that the body deformation depends not only on the motion of the underlying skeleton but also on the properties of soft tissue and tendons. Thus, soft tissues and tendons should be integrated into the model.

Both approaches have several imperfections, but, it can be also seen that some of these are particular to one of the methods and are compensated in the other method. This suggests that using a combined approach would benefit from the fast model prototyping and intrinsic accuracy of parametric layered-models and from the expressiveness of the explicit models.

2.1.3 Character Modelling in Modern Animation Systems

In the last decade, Character Animation became an integral part of Computer Animation, so it is no wonder that all major animation systems have implemented tools for modelling and animating characters. This section gives an overview of these tools.

The most common way to model a character is to employ user-defined hierarchical skeletons (Figure 2.6) and deformable "skins". Skeletons are based on a basic feature of any 3D-animation package – object linking, which allows child objects to inherit transformations of their parent objects. The animation software automates the creation of such hierarchies and simplifies their controllability by providing Inverse Kinematics apparatus. The complete model is created by skinning a skeleton i.e. attaching a mesh or NURBS model to the skeleton using the algorithm outlined in section 2.1.2.

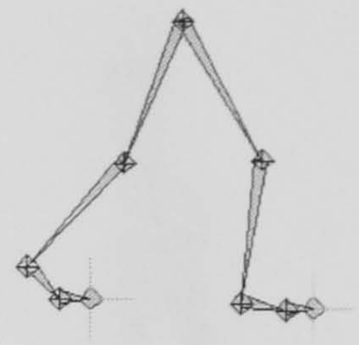


Figure 2.6: Bones system in 3DS Max

This is a general-purpose approach. It allows the creation of characters of different complexity and topology. But, this also means that such models are poorly compatible with some advanced animation techniques (including Motion Capture) that have been developed for use with specific characters (bipeds, for instance). To overcome this problem some animation packages provide special tools designed to model and animate a particular type of characters. The most powerful of them are *Poser* [Poser Review, 1999] and the *Character Studio* plug-in for *3D Studio Max* [Character Studio User Guide, 1998].

In contrast to other 3D-animation systems, Poser is exclusively designed for modelling and animating characters and has impressive parametric-modelling capabilities. It has a wide range of ready character models (human, animals, fairy characters, etc); these support many high-level parameters that can be modified in order to obtain the desired results. Needless to say, the software allows cloth and hair to be added to the models.

Another strong point of Poser is its posing and facial-expression control functionalities. A user can manipulate the pose and the facial expression of the model not only by means of standard low-level transformation tools (rotate, move, etc) and Inverse Kinematics, but also using high-level controls like "straighten the torso" or "smile".

Unfortunately, the animation part of Poser does not match its strong modelling and posing capabilities. This, and the weak support of non-character objects, make the program more suitable for the creation of static images (Figure 2.7) or simple character-only animations rather than for creation of complex animations.

Figure 2.7: Posing a character in Poser is easy

Unlike Poser, Character Studio specialises in animation rather than modelling. It does not have such extensive character creation capabilities as Poser, however it does provide a strong set of character-animation tools including character-specialised Inverse

Kinematics, Motion Capture, Motion Blending and IK-based procedural tool for creating walking and running locomotion. Its modelling part consists of a predefined skeleton model (Figure 2.8) and a very powerful skin-deformation plug-in Physique.

Figure 2.8: Character Studio's Biped

Thanks to the object-oriented architecture of 3D Studio Max most of its tools can be shared between different parts of the system. For instance, Character Studio's Physique modifier is used to 'skin' our figure model implemented as a plug-in for 3D Studio Max. Thus we could concentrate on the animation content of the model leaving the visualisation and character-modelling functions to the system.

2.2 Animating the Model

This section discusses the principal methods used to animate skeleton models and examines a practical approach to character locomotion implemented in 3D Studio Max. Before starting to describe the methods, some evaluation criteria should be introduced so that one can compare the methods to each other and identify their drawbacks.

- *How automatic is the method?* An ideal method should generate a motion automatically; the user just selects what kind of motion is required and specifies a few high-level parameters. Also, only automatic methods can be used to create animations for real-time applications (VR, real-time TV shows, simulators, etc).
- *Is it easy to control the method?* A method that has non-intuitive control (for example a control that requires a user to specify muscle forces producing the motion) or requires an animator to have special skills, cannot be considered as a very usable method.
- *Does the method require extra equipment (video cameras to capture a motion, etc)?* Generally it is better if the method does not need such equipment. Though in some cases using special equipment can be the only option (for instance in a live TV show).
- *Is the method general-purpose i.e. can it be used to generate different kind of animations (simple walking, walking on uneven terrain, running, walking up/down staircase, etc)?*
- *And finally, does the animation produced by this method look natural?* This is the most important, but also the most difficult criterion to evaluate.

2.2.1 Keyframe Animation

Keyframe Animation is one of the most widely-used animation techniques. The concept of this method, as well as its name, was taken from traditional animation; the idea to divide frames to key and in-between frames was originally introduced in the hierarchical animation-

production system developed by Walt Disney. Professional animators draw keyframes that define the animation with less skilled animators drawing the in-between frames.

Applying this concept into the context of Computer Animation means that an animator defines the keyframes and the animation software interpolates the other frames.

Even though this method can dramatically reduce the number of frames that have to be created manually, it is still quite time consuming. In order to improve the method, many developments have been made [Nebel, 1999; Steketee, 1985]. Nowadays, animation programs provide extensive possibilities to control the process of creation and editing of keys and to control the interpolation of in-between frames.

The use of the keyframe technique for Character Animation means that an animator sets key-postures (i.e. specifies joint angles) of a figure and the algorithm interpolates in-between postures. The keys correspond to the postures that are characteristic for a particular type of motion. For instance, the key postures for a walking motion should include heelstrike, midstance and toe-off positions.

It should be stressed how demanding the work of setting the key-postures is. While the positioning of independent objects is quite straightforward, the positioning of hierarchical structures, in which a transformation of a parent object leads to transformations of all its children, is extremely tedious. To help animators, the latest versions of animation software usually allow Inverse Kinematics to be used to position hierarchical objects.

Besides the traditional approach in which an animator manually sets postures, a mixed keyframe–procedural approach can be used. A procedural algorithm is used to create key-postures; then the animator tunes these keyframes and possibly adds some extra keyframes to avoid interpolation artefacts such as penetration of the ground by the feet.

Interpolation artefacts are a common problem in all animation methods that do not apply constraints to the motion. They make the keyframe animation more difficult and time-consuming because the animator has to create extra keyframes, the only purpose of which is to remove artefacts. Another way to fight these artefacts is to use an "intelligent" interpolation controller that can process constraints.

The idea of keyframes is fundamental to Character Animation and it is constantly used throughout this chapter. Moreover, some techniques described below can be considered as keyframe techniques despite being considered in another section. The reason for this is that

the borders between different animation techniques are often unclear; so, for each method an attempt was made to select the new ideas only and not to repeat the basic ones.

2.2.2 Animation Based on Motion Capture

Nowadays the animation market needs an enormous amount of high-quality character animation. However, it would be impossible to satisfy these demands using traditional keyframe-based methods only. Even a professional animator needs many days of tedious work to create a minute-long clip of character animation. Unfortunately, simulation methods, which can significantly reduce the amount of manual work, are still not powerful enough to be a real help for animators. Thus, the only alternative solution is to use Motion Capture.

Data Collection Phase

The idea of MC-based methods is to 'capture' movements of a real actor and map these movements to a computer character. This procedure can be divided into two distinctive phases: the data-collection and the animation. The first phase involves collecting and pre-processing the data, which is then used for animating a model(s) in the second phase.

Though, the term Motion Capture is often used to refer to both, the data collection and the animation phases, this is not strictly correct: it should be used regarding the data collection only. More precisely, Motion Capture can be defined as a process of recording a live motion event and translating it into a usable mathematical representation of the performance.

There are several approaches to capturing the motion, the main ones being optical, magnetic and mechanical methods. These names reflect the type of device used for data collection; corresponding to video cameras, electro-magnetic sensors and mechanical potentiometers. Each of these methods has specific drawbacks and benefits, which should be taken into account when choosing the method for a particular task.

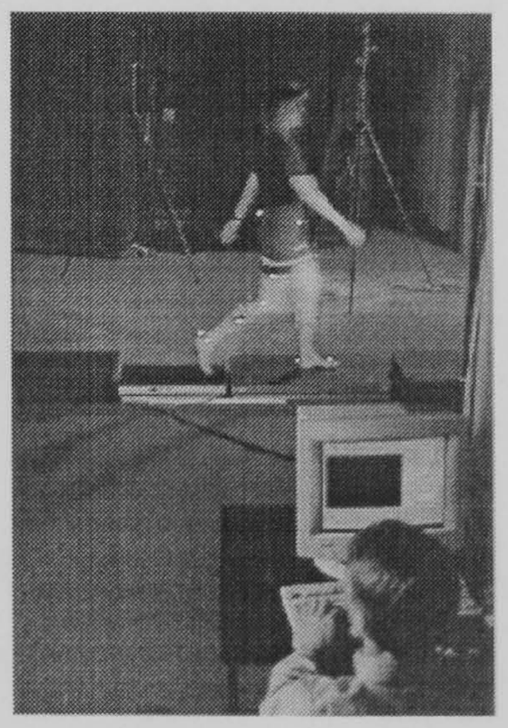


Figure 2.9: Capturing a performance using optical MC system

In *the optical MC system*, several video cameras are used to record different views of the performance. Then a computer analyses the video stream, recognises reflective markers on the actor's body (Figure 2.9) and reconstructs their 3D positions by combining the information from several cameras. The result of these calculations is a set of marker coordinates. But since many animation packages are able to deal with joint-rotation data only, the motion capture system performs an additional step extracting rotations from the translational data.

A lot of processing has to be done to extract the motion from a raw video stream. The first major step is to find and identify the markers. If detecting a marker on the image is not difficult, differentiating visually indistinguishable[1] markers from each other is a challenge. The corresponding algorithms are based on continuity of marker trajectories (i.e. if we know marker positions for frames *i-n* .. *i* then we can extrapolate its position at frame *i+1*). Though these algorithms are constantly improving, they are still not 100% fault-free and therefore operator intervention may be required to assign markers in difficult situations (occlusions, etc). The second step is responsible for reconstructing the 3D positions of the markers. A single image from a calibrated camera provides enough information to estimate a line on which the marker lies and, by combining information from several images, the algorithm triangulates the exact 3D position of the marker.

At the time being, optical-based MC is the most common MC technology. The reasons for this are:

- high accuracy of the optical data;
- absence of hardware limitations on the number of markers or actors;
- performance is not limited to a small area (provided that one can afford extra equipment);
- movements of the actors are not limited by cables or any mechanical devices.

Among the drawbacks of optical MC, one can number the following:

- Optical MC is not a real-time method and it requires extensive post-processing. Moreover, the process of marker tracking is not fully automatic and, occasionally, operator intervention is needed to resolve ambiguities;
- Equipment is very expensive. Only large animation companies can afford to have an in-house studio. This has lead to the establishment of many companies that specialise in providing a MC service;
- Optical motion capture is very sensitive to the calibration process. If one of the cameras is misaligned (due to calibration error or a movement of the camera) than the measurements from different cameras will be inconsistent, making it very difficult to track markers and calculate their positions.

A viable alternative to optical systems is *magnetic MC*. Magnetic systems consist of transmitters that create an electro-magnetic field in the performance area and a set of sensors

---

[1] Some systems support plainly distinguishable markers (for instance, active light-emitting markers). However, using such markers requires advanced, and therefore more expensive, hardware.

attached to the performer(s). Each sensor is able to detect the power and direction of the signal from each of the transmitters and to send this information to the workstation. The computer processes the data from the sensors to determine their positions and orientations.

Since this procedure is fully automatic and does not need much computational power it can be performed in real-time, which makes magnetic MC systems particularly useful for online applications.

In comparison with the optical systems, magnetic systems have rather more limitations:

- range of the performance areas is smaller than the ranges of video-based systems;
- the performer's movements are limited by cumbersome equipment (wires) – though some modern systems use radios to transfer the data from sensors to the computer (for instance, Motion Star Wireless by Ascension Technology, Figure 2.10);
- the accuracy of these systems is generally lower then the accuracy of optical and mechanical systems, and their sensitivity to electro-magnetic interference contributes to the problem.

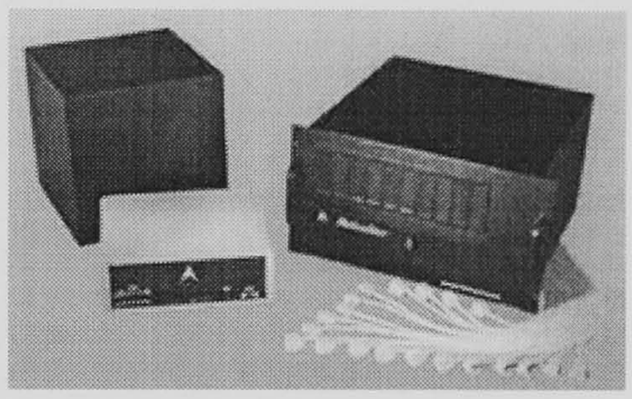


Figure 2.10: Magnetic MC system *Motion Star Wireless*

The third method to capture motion data is to use electro-mechanical devices that measure joint angles (goniometers). One of the most popular devices of this kind is a potentiometer-based goniometer. The principle is simple: it has a "slider" whose movements change the strength of a variable resistor. Thus by measuring the resistance, one can gauge the amount of movement.

Basically, one potentiometer can only measure one degree of freedom (translational or rotational) so to measure all possible transformations in a joint, one has to use six devices. Analogously, several goniometers can be combined in an exoskeleton-like structure to track full-body movements.

Like the magnetic systems, goniometers can capture joint rotations directly, by-passing the intermediate use of translational data. This not only improves their accuracy but also makes the measurements very fast. These advantages make goniometers a natural choice if high accuracy or real-time feedback is required.

Though mechanical devices are very useful for tracking local motions, such as head turning or hand articulations (cyber gloves), their use for capturing full-body motion is very limited. The

reason for this is the extreme awkwardness of full-body tracking systems (exoskeletons) that significantly limit the freedom of movement. Consequently, mechanical exoskeletons were replaced by other MC systems, optical and magnetic, leaving exoskeletons to be used only in specific niches, such as some Virtual Reality applications.

Finally, there is another approach to capturing character motion, which has become a popular topic of research in the last few years. This is *vision-based* MC. Its idea is to use computer vision algorithms to extract motion information from a normal video stream [Aggarwal, 1999; Davis, 1998]. Though the use of this approach in some specific areas of Computer Graphics (for instance, motion capture of face movements) looks very realistic and promising, its use for accurate capture of various figure movements does not seem feasible in the near future.

No matter which device is used to capture the performance, the resulting data should come to the animator in some unified form. Since the data is most likely to be used to animate a skeleton model, they should provide rotational data for joints and translations for the root link. These may be also accompanied by marker coordinate data (if optical or magnetic methods were used). Since the rotations are defined relative to some base position, this position should be provided with the data, too.

The accuracy of motion-capture data is defined by several factors. The first factor is the measuring error of the capturing device. The second factor is the error originating from the algorithm that maps the 3D marker positions to the joint rotations. This error is introduced due to the variance between the motions of markers and the bones, and due to the discrepancy between a real skeleton and a simplified synthetic skeleton. The mechanical devices such as goniometers are affected by these problems, too: such devices are attached to the body by straps and therefore they are also affected by skin slipping and muscle deformation. Another potential source of errors is the specification of the base position: if the base posture used in the animation system is not close enough to the base posture of the performer then the reconstructed motion can seriously deviate from the original.

Accumulation of joint-rotation errors in the skeleton hierarchy can result in a significant error in the position of the end-effectors (feet, toes, and hands). To reduce this error special optimisation methods can be applied. These will reduce the difference between the original and the synthetic trajectories of end-effectors by means of Inverse Kinematics techniques [Choi, 1999] or using some optimisation techniques.

Animation Phase

Once the motion data has been captured and preprocessed the animation phase starts and the data are used to animate a skeleton model. Not only are there many ways to capture the data, there are also many ways to use it. In the simplest case, the data is just replayed as it is. If the performer and the model have the same proportions, then this method will reproduce the original motion. Though this approach may be adequate for some real-time applications, most applications need the motion to be modified in order to create the required animations. This is achieved by means of various motion-editing techniques.

These can be divided into two major groups. The task of the first group is to adapt the captured motions to different models and to take into account the interaction with the synthetic environment. The methods from the group are often referred to as Motion Retargetting or Motion Correction methods. If these methods try to preserve the characteristics of the original motions, the second group of techniques aims to create *new* motions by blending and modifying motions from a database. These are Motion Blending and Motion Warping methods.

Motion Retargetting Problem

It is uncommon to have a character that is an exact replica of the performer. Consequently, the animators often encounter the motion retargetting problem, that is, when the deviation between the sizes and proportions of the character result in the motions violating constraints or not meeting the original goals. The manual correction of these motions can be very time-consuming, which obviates one of the main advantages of MC approach – its efficiency. That is why there is a high demand for automatic motion-correction methods.

At the present time, two basic approaches are used to adapt a motion captured from one character to another character. The first gives priority to the original end-effector trajectories and modifies the joint data to retain these trajectories. In contrast, the second approach is to try to preserve the angular joint data while putting constraints on the end-effectors.

The first method is very similar to the accuracy-enhancing technique described above. Here, however, the target and the original models are different, and so are the relative positions of the end-effectors. This can make it impossible to maintain several trajectories together. For instance, the trajectories of the toes and the heel of the same foot cannot be followed at the same time if the length of the foot has changed.

Choi et al. [Choi et al., 1999] performed a real-time retargetting of various motions, including walking and bat-swinging motions. They used Inverse Kinematics to maintain the original trajectories of the toe-tips (walking motions) and of the hand (bat swinging motion). Though the algorithm succeeded in its task, the changes it introduced to the style of the motion were quite serious, particularly to the style of bat swinging motion.

As a rule, IK-based correction algorithms deal with underdetermined problems, i.e. tasks where the number of constraints is fewer than the number of free degrees of freedom. Theoretically, this means that an infinite number of solutions exist for any such problem and special criteria should be used to choose the most appropriate one. In character animation, choosing the right criteria (or *a secondary task* in IK terms) is a very important part of any IK method, which explains the existence of many different IK-based methods. A typical problem associated with the indeterminacy of an IK task is discontinuity of its solution, which may result in jerky motions. For instance, Choi reported that the direct application of their algorithm produced jerking of the pelvis; to solve this problem, they imposed additional constraints on pelvis motion.

Indeterminacy of the task is not the only problem of this approach. As mentioned before, the style of the motion can be seriously altered by the correction and in some cases, this can be unacceptable. Also, these methods are based on the assumption that the original end-effector trajectories are correct, so it cannot be used if the scene is different from the performance environment.

A more flexible and powerful approach to the retargetting problem is based on use of general kinematic and dynamic constraints. This allows the end-effectors to move away from the original trajectories but guarantees that their new trajectories, and the joint motions, meet specific constraints.

If one chooses to use this approach, the first problem one encounters is the definition of the constraints. In the previous method, the constraints (i.e. exact 3D positions of the end-effectors) were explicitly defined in the motion data. Here, however, some constraints (or their status) may be unknown in advance and they must therefore be specified by the animator or by the program. An efficient method should identify most of the constraints automatically, leaving the animator to specify only a few goal-specific constraints.

Foot-ground constraints are likely to be the most common class of constraints used in Character Animation. They are divided into several types in accordance with the type of foot-ground interaction and the structure of the model: foot above the ground, heel contact, flat

foot contact, etc. Only one of these can be active at any time (for one foot), and the type of the active constraint can change from one frame to another. Obviously, manual specification of these constraints is a time-consuming process and it would be very useful if the animation program could detect the active constraints automatically. Unfortunately, this is not a straightforward process and so far such a detection algorithm has not been fully implemented in animation[1]. The only attempts to identify the state of the feet during locomotion were undertaken for simplified "footless" models. These algorithms used bounding-box criteria [Rose et al., 1996] to determine when the foot is in the contact with the ground. The other methods either relies to manual specification of the constraints [Gleicher, 1998] or use simplified foot-above-the-ground constraint [Boulic, 1992].

On one hand, the introduction of general constraints complicates the motion retargetting, but on the other hand, it provides greater flexibility and may better preserve the original character of the motion. Consider retargetting a walking motion to a smaller character; if the algorithm retains the original trajectories of the feet, then the small character will perform unrealistically long strides (Figure 2.11). Alternatively, the algorithm may try to preserve the joint trajectories and adapt the translational motion of the character to ensure that the feet are not slipping; this approach is more likely to retain the original character of the motion.

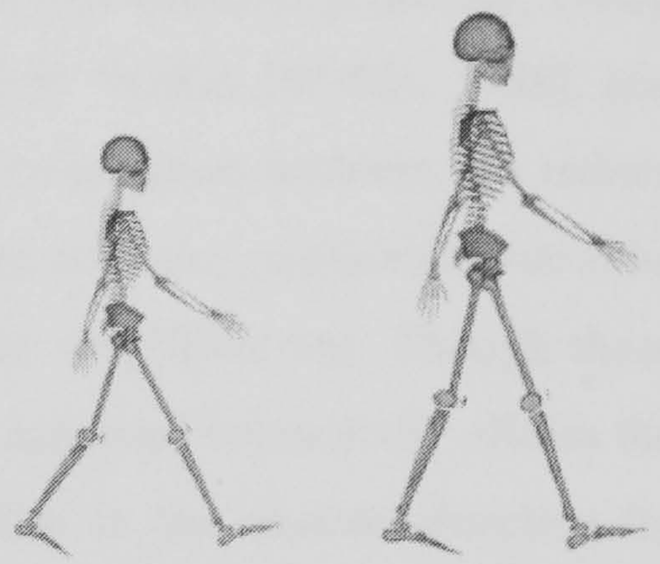


Figure 2.11: Same toe positions - different figures.

This is a simple example. In reality, retargetting algorithms have to deal with goal-oriented motions where a character interacts with some other characters or objects located in the scene. For instance, the character in the previous example may have a goal to touch an object located at some distance. If the recorded motion is simply applied to a smaller character, then this character will walk a shorter distance and will not be able to reach the object. Such problems can be corrected by incorporating goal constraints.

Processing complex goal-oriented constraints makes the task of retargetting much more difficult. The method that is often used here is a constrained optimisation, which employs numerical iterative algorithms to calculate a motion that satisfy the constraints. In this case, the original motion data are used as a starting point for the optimisation.

---

[1] Biomechanics people are also interested in extracting similar information from motion capture data and therefore they try to implement such functionalities into motion analysis tools.

Constrained optimisation is a powerful mathematical technique but it still cannot find an expected solution in a reasonable time unless it is helped by heuristic algorithms and it is guided by user-defined intermediate constraints. Clearly, no algorithm or a set of heuristics can work for every motion. A possible way out of this problem might be the integration of a retargetting algorithm into an interactive animation system. Such system should:

- be fast enough to provide interactivity;
- be able to decompose motions into parts: shorter and simpler motion sequences can be processed quicker. Also there are more chance to find an appropriate heuristic solution if a motion is simpler;
- support intuitive control over constraints. It would be a benefit if the system can also detect some of the constraints automatically.

A prototype of an interactive retargetting system was developed by Gleicher [Gleicher, 1998]. In this system, he used spacetime constraints, first proposed by Witkin [Witkin, 1988], and sequential programming algorithm to solve the constrained-optimisation problem. To reduce the computational complexity of the algorithm, and to give the animator additional control he had to sacrifice dynamic constraints and to make some other simplifications. Though these can result in non-optimal or physically incorrect motions, the achieved interactivity allows the user to monitor the computations and to guide the algorithm in the desired direction by making adjustments to the constraints.

Optimisation techniques are not the only methods that can be used for constraint-based retargetting: traditional IK is another common technique used for this purpose. A typical approach is to combine both direct and inverse kinematics i.e. using the original motion when the constraints are not violated, and employ IK correction to enforce the violated constraints.

This approach was studied by Boulic [Boulic et al., 1990] who called it the Coach-Trainee method. The main idea of their method was to consider the original joint trajectories as a reference motion to be put into the secondary task of the IK algorithm. Thus, the main IK task guarantees the satisfaction of the constraints, while the secondary task ensures that the resulting motion (trainee) is close to the original one (coach). To maintain the continuity of the motion, the authors also introduced a transition function whose purpose was to interpolate the transition between the corrected and the uncorrected (original) motions.

The nature of the IK algorithm makes it poorly suited to large motion modifications, which can be required to edit complex goal-oriented motions. However, it can still be very useful for many simpler motions, particularly for basic motions used in motion libraries.

Motion Editing

For many applications, MC-based animation is a viable alternative to traditional keyframe animation. Yet, its use is not often justified because of the costs, financial and manpower, associated with capturing the data. A practical solution to this problem will be the use of libraries with pre-recorded motions. This would not only allow more animators to use MC, but it would also make it possible to utilise real motion data in applications for which a capture-on-demand approach is not suitable (for instance, simulators, games, etc).

Of course, no motion library can provide an animator with all the motions that she/he needs. But what such library can do is supply a variety of basis motions, which can be edited and pasted together to provide a desired motion. Therefore, special motion-editing tools are needed to take a full advantage of motion libraries.

The development of these tools goes in three major directions:

- Motion Editing: altering basis motions to impart desired characteristics to them. The techniques employed here are Motion Warping, which is used to simplify and automate modification of single motions, and Motion Blending, which create motions by combining (interpolating) several basis motions;
- Motion Montage: creating complex animations by sticking basis motions together. One of the main tasks here is automatic generation of realistic transitions between the motion segments;
- Motion Retargetting and Correction: dealing with adapting motions to new characters and enforcing constraints.

Motion Blending methods aim to produce a motion with desired characteristics by interpolating the parameters of basis motions, i.e. pre-recorded motions from a library.

Which parameters are interpolated depends upon the algorithm and the type of motion (walking, dancing, etc). These can be the joint-motion trajectories, defined either in the frequency or in the temporal domain, or they may be 3D positions of the joint centres or some other control points.

Before the interpolation can be performed, the source motions have to be synchronised; for instance, two walking motions should have their characteristic points (heelstrikes, etc) matched. For short non-periodic motions, this synchronisation can be done interactively, whereas periodic motions like walking or running should be synchronised by the program.

One of the methods for synchronising two motions is a non-linear signal-matching procedure, adopted from speech recognition. This procedure identifies transformations (compressions and expansions of the time) that best match the signals. The algorithm already gives control over the speed of the motion: the transformation function can be used to map the speed of one basis motion to the other motion or to interpolate between their speeds. But, more importantly, it solves the problem of synchronisation, making it possible to perform a meaningful interpolation of the amplitudes, the second component of the signal.

Below is a flow chart of a typical motion-blending tool:

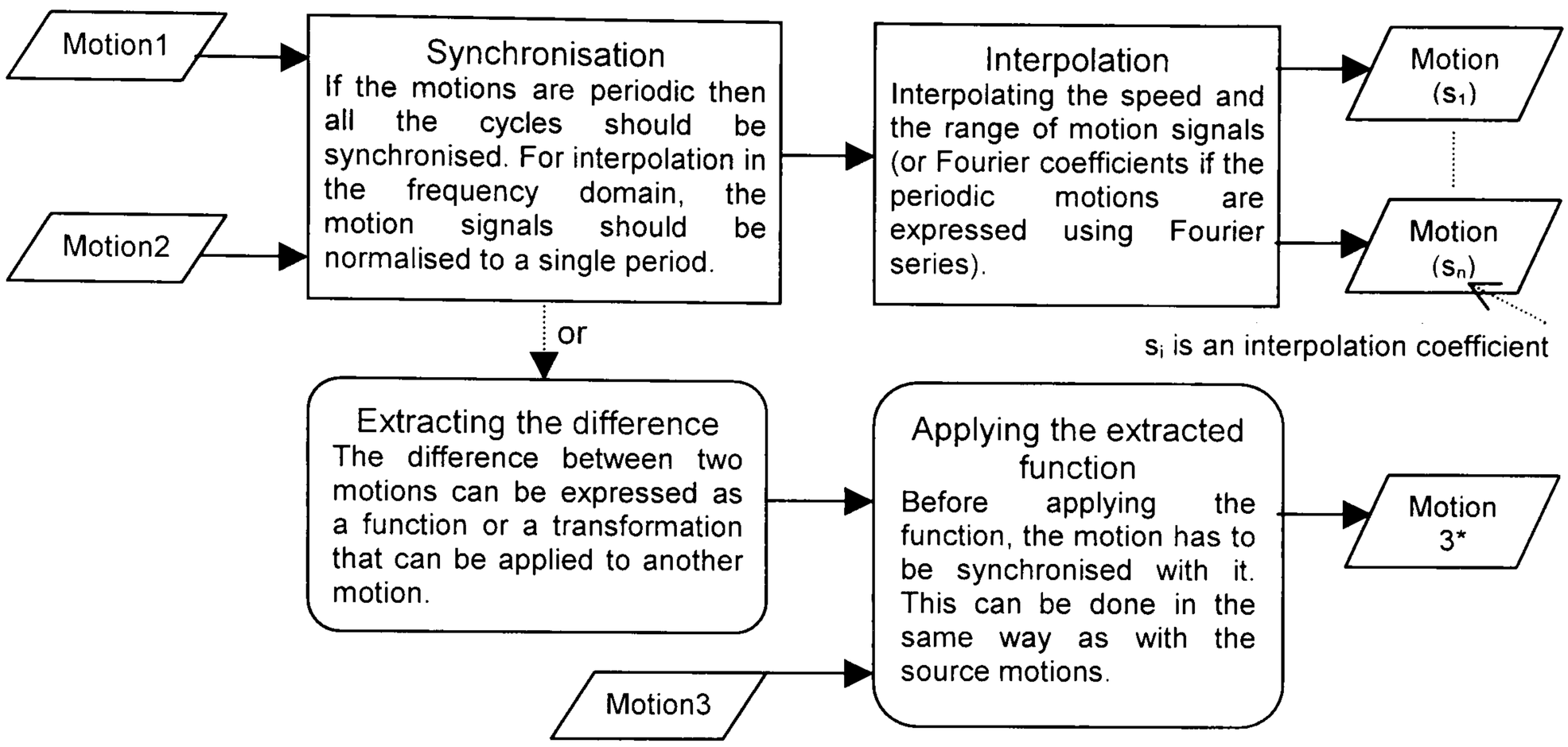


Figure 2.12: A flow-chart of a motion blending algorithm

Example (from the experiments made by Unuma [Unuma et al., 1995]):

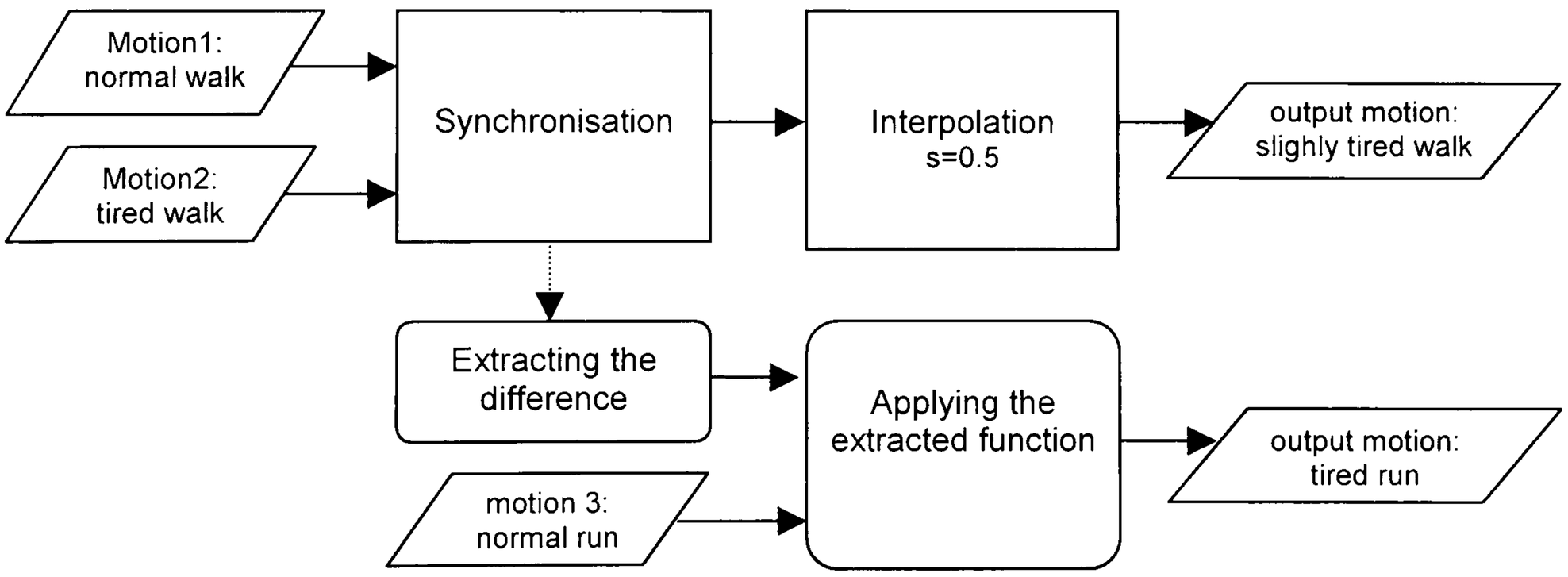


Figure 2.13: Example of data flow in Unuma's motion blending algorithm

Motion Blending is primary an empirical technique: currently there is no theoretical basis to allow the correlation of specific characteristics of the motion to particular interpolations. Nevertheless, numerous experiments with motion-blending tools have resulted in the creation of an "empirical knowledge base" that can suggest a possible result of some interpolations,

i.e. which interpolations work and which do not. Moreover, functioning transformations that modify particular characteristics of the motion (emotions, for example) can be extracted and used autonomously.

Among experiments on motion blending, one should mention studies on manipulating emotional characteristics of human motions. Various research-groups [Unuma et al., 1995; Amaya et al., 1996] have investigated the possibility of interpolating between motions with different emotional tints. These studies suggest a correlation between the speed/amplitude of joint trajectories and the emotional characteristics of motion and propose to control these characteristics by interpolating between several basis motions such as normal, angry, tired, etc.

Motion Warping represents another group of techniques designed to reuse captured or keyframe animations. As the name suggests, these techniques warp (edit) the original motion signals to produce a new motion.

There are many reasons why one may need to edit a motion: to match the motion with the environment or with the movements of other characters; to correct artefacts; to alter the character of the motion, etc. Some of these tasks require local modifications but some imply modification of the whole motion. These means that a warping algorithm should be able:

to make local adjustments to motion signal so as the edit segments fits naturally into the motion

to make global modifications efficient i.e. without having to modify every frame or keyframe

In their work, Witkin and Popovic [Witkin et al., 1995] proposed to adapt the traditional keyframe technique for motion editing needs. The user can interactively modify postures at few keyframes. Then, the algorithm applies a smooth transformation to the signal; this transformation does not change important details of the signal but enforces user-defined keyframe constraints. The smoothness of the transformation is guaranteed by using splines:

$t = g(t')$ - timewarp transformation.
$\theta'(t) = a(t) \bullet \theta(t) + b(t)$ – scale-and-shift transformation of the signal, where
$g()$, $a()$, $b()$ – Cardinal splines.

An obvious advantage of this approach is that it easily fits into any keyframe-based system (in fact, the authors built their algorithm into *Softimage*). On the other hand, the keyframe origin of the approach means that it inherits the drawbacks of keyframe animation; for instance, the difficulties of enforcing constraints between the keyframes.

Instead of working with the original trajectories, one can work with trajectories produced by applying multi-resolution filtering to the signals [Bruderlin et al., 1995; Sun et al., 1997].

The main advantage of using multi-resolution filtering is the possibility to choose from several levels of detail (Figure 2.14) [Finkelstein, 1994]. For instance, if the animator wants to change the general pattern of the motion, he/she can alter the low-frequency approximation. Alternatively, to change small details, the animator should edit the high-frequency part of the signals.

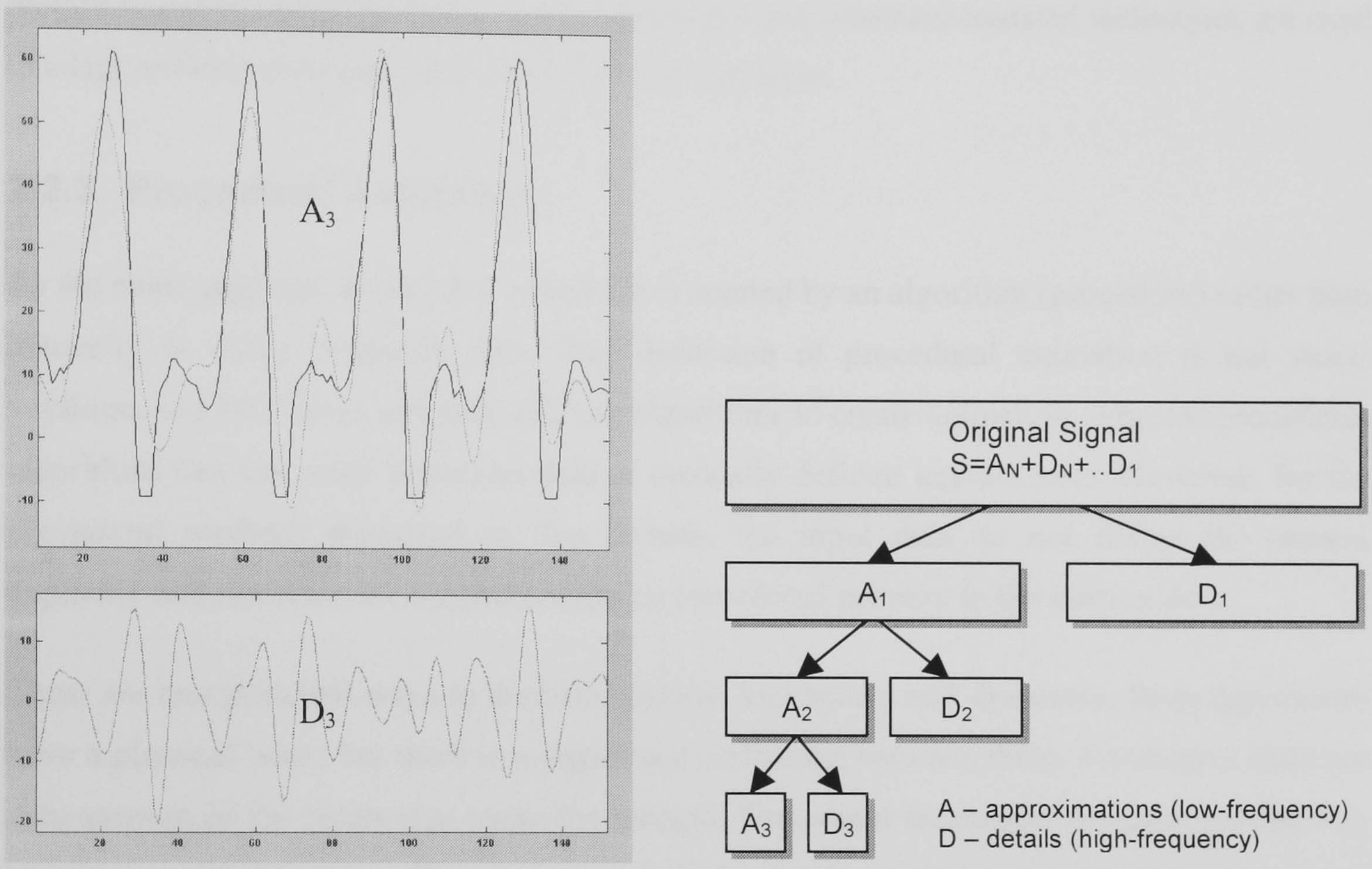


Figure 2.14: Wavelet decomposition of hip-flexion trajectory (original and its approximation [A_3] - top graph, the correspondent high-frequency signal [D_3] - bottom graph).

Despite the efficiency of multi-resolution editing and analogous trajectory-editing methods, they are still not very useful as standalone tools. Curve-based representation of the motion (or any other mathematical representation) is difficult for human perception and it is hard to obtain a desired motion by manipulating with curves. Hence, these methods should be used together with another animation technique, and this technique should be either automatic or should have human-friendly controls.

One can see from this review that a lot of effort has been invested in developing MC-based animation techniques. But if one looks at commercial animation packages, one will not see many of these advanced techniques implemented in end products. This is not surprising: animators need general solutions that can handle a wide set of problems. They are not interested in programs that can only deal with a limited set of motions (cup grasping, for example), even if they do it perfectly.

This situation is constantly improving. Motion-editing techniques are becoming more reliable and more universal. Still, no single method can solve all the problems alone and the only way to go is to integrate the various methods into one framework.

While advanced MC tools remain research prototypes, the creation of high-quality animations is performed semi-manually. The performance is designed to be as close as possible to the target motion; and once the performance has been captured, the motion is manually adjusted to produce the desired result. As for less-demanding applications, the situation here is better: motion re-use is becoming more widespread, and some computer-assisted techniques are used to adapt motions or to generate transitions between them.

2.2.3 Procedural Animation

As the name suggests, procedural animation is created by an algorithm (procedure) rather than manually or using measured data. This definition of procedural animation is not strict: keyframe and MC-based methods also use algorithms to create animation, whereas procedural algorithms can use some measured data or manually defined key-postures. However, for the procedural methods discussed in this section, the input data do not define the motion explicitly and therefore the algorithms can be considered primary to the motion data.

There are two principal ways to describe motion: kinematics and dynamics. Both approaches have a physical basis, but there is a significant difference between them: kinematics does not take account of the forces that cause the motion. Procedural techniques are also divided into kinematics and dynamics. The kinematics-based procedures deal with time, distance, speed and acceleration; while the dynamics algorithms additionally use moments and forces to reconstruct the motion.

Whatever algorithm is used, it cannot create an animation without the motion being defined. In keyframe and MC-based animation, motions are defined explicitly, by specifying values of each DOF at every frame (keyframe). In procedural animation, the situation is different. Here, the motions are defined indirectly using goals (goal-directed approach) or using parameterisation (parametric or knowledge-driven approach).

The idea of a *parametric* approach is to create a parametric model of some motion and use it to generate required motions that are of the same kind. For instance, a kinematic walking model can be parameterised to generate walking motions with different speed, stride length, etc. Parametric methods are faster and more reliable than goal-directed methods, which makes them more suitable for real-time applications (VR, games).

In the *goal-directed* approach, the motion is defined implicitly using one or several goals, which have to be achieved in the motion. The algorithms work by searching for motions that satisfy goal-constraints and some secondary criteria. Since the constraints are usually put on end-effectors not on joint rotations, the goal-directed algorithms are often based on inverse methods (inverse kinematics or inverse dynamics). Except for a few unusual methods[1], goal-directed methods cannot produce the desired motion using goals only. Normally they use either a parametric model or captured data to help the algorithm to find the solution.

The goal-directed approach was considered earlier in the discussion of Motion Retargetting in section 1.2.2. Using the captured performance as an initial estimate of the desired motion, the retargetting algorithms apply optimization techniques to find the desired motion. Analogous techniques are used to incorporate goals into procedural methods.

Parametric and Knowledge-Driven Methods

As noted above, the kernel of parametric and knowledge-driven algorithms is a model of motion. This model can be based on either theoretical or empirical knowledge about the motion. An example of purely theoretical approach is a dynamic simulation of a bouncing ball: the motion is fully defined by physical laws and the dynamic properties of the ball and the surface. In Character Animation, though, a fully theoretical approach is of hardly any use. This is because human locomotion is very complex[2], and it is practically impossible to create an accurate physical (kinematics or dynamics) model of locomotion. Therefore the algorithms used in Character Animation are either completely empirical or combine the empirical and theoretical knowledge.

Though biomechanics does not provide explicit knowledge of how to simulate human locomotion, it provides many descriptive models that can be used for reconstructing the locomotion. These models include descriptions of the locomotion phases, experimentally-derived formulas for various parameters of the locomotion, etc.

---

[1] There is a family of methods that starts the motion-search process from scratch, i.e. without using any parametric or captured motion as an initial estimate. These methods mainly use genetic algorithms to find the motion that realises the given goals. Owing to the random nature of the algorithm, motions produced by these methods are never realistic (though they are often very amusing).

[2] Since the complexity of locomotion models is mainly due to foot-ground contact, one can easily create dynamics (or even kinematics) models for locomotions that do not involve foot-ground interaction (jumping, falling).

The actual way the motion is defined depends upon the particular implementation. In the earliest models, walking and running animations were created by interpolating between few key-postures set by the algorithm. This approach was first proposed by Zeltzer [Zeltzer, 1982]. He incorporated biomechanical knowledge of walking into a finite-state machine that created key-postures. This finite-state machine was controlled by high-level parameters such as stride length and cadence. The final animation was produced by linear interpolation between key-postures.

Since Zeltzer's algorithm did not consider ground constraints, it could not guarantee that the feet would not penetrate the ground. A simple way to avoid this problem was proposed by Bruderlin and Calvert [Bruderlin et al., 1989; Bruderlin et al., 1993]. The trick is to put the root of the skeleton at the current support foot and to switch the root to the other foot after each foot strike. Then the positions of the proximal links (including the pelvis) are calculated from the state constraints. On the one hand, this approach guarantees that the stance foot will not slide or move up/down during the stance phase. On the other hand, this approach may result in artefacts in other aspects of the motion, such as jerky motion of the pelvis and torso.

Another way of avoiding artefacts associated with blind interpolation of key-postures is to use Inverse Kinematics (see Appendix A). The idea of using IK to calculate the joint angle trajectories from the trajectories of the end-effectors (feet) was first proposed by Girard and Maciejewski [Girard et al., 1985]. Their animation system PODA used IK to reconstruct the joint angles from the animator-defined trajectories of the feet and the body. Initially this method was closer to keyframe methods than to the procedural methods, but its further development resulted in the emergence of one of the most remarkable procedural animation algorithms: *a footstep-driven algorithm.*

In the footstep-driven algorithm, a finite state machine uses some heuristic formulas and high-level parameters (speed, number of steps, direction, etc) to create footsteps. These footsteps, which can also be positioned manually, are used as constraints for the IK algorithm that generates the angle trajectories. The footstep-driven algorithm is one of the few procedural methods that has been implemented and used commercially, as a plug-in for 3D Studio MAX (Section 1.2.4).

The IK approach was also picked up by other researchers. Boulic [Boulic et al., 1992] used IK to assert constraints in their procedural algorithm for animation of walking. This algorithm was based on a kinematic walking model built from experimental data and consisting of a set of parameterised trajectories (Figure 2.15) for all DOFs. The user could interactively adjust high-level parameters of the model, such as velocity and cadence, and then the algorithm

would produce joint-motion and body position trajectories. The IK-based correction method (*Coach-Trainee*) was applied to generated motion to guarantee the foot-ground constraints are not violated.

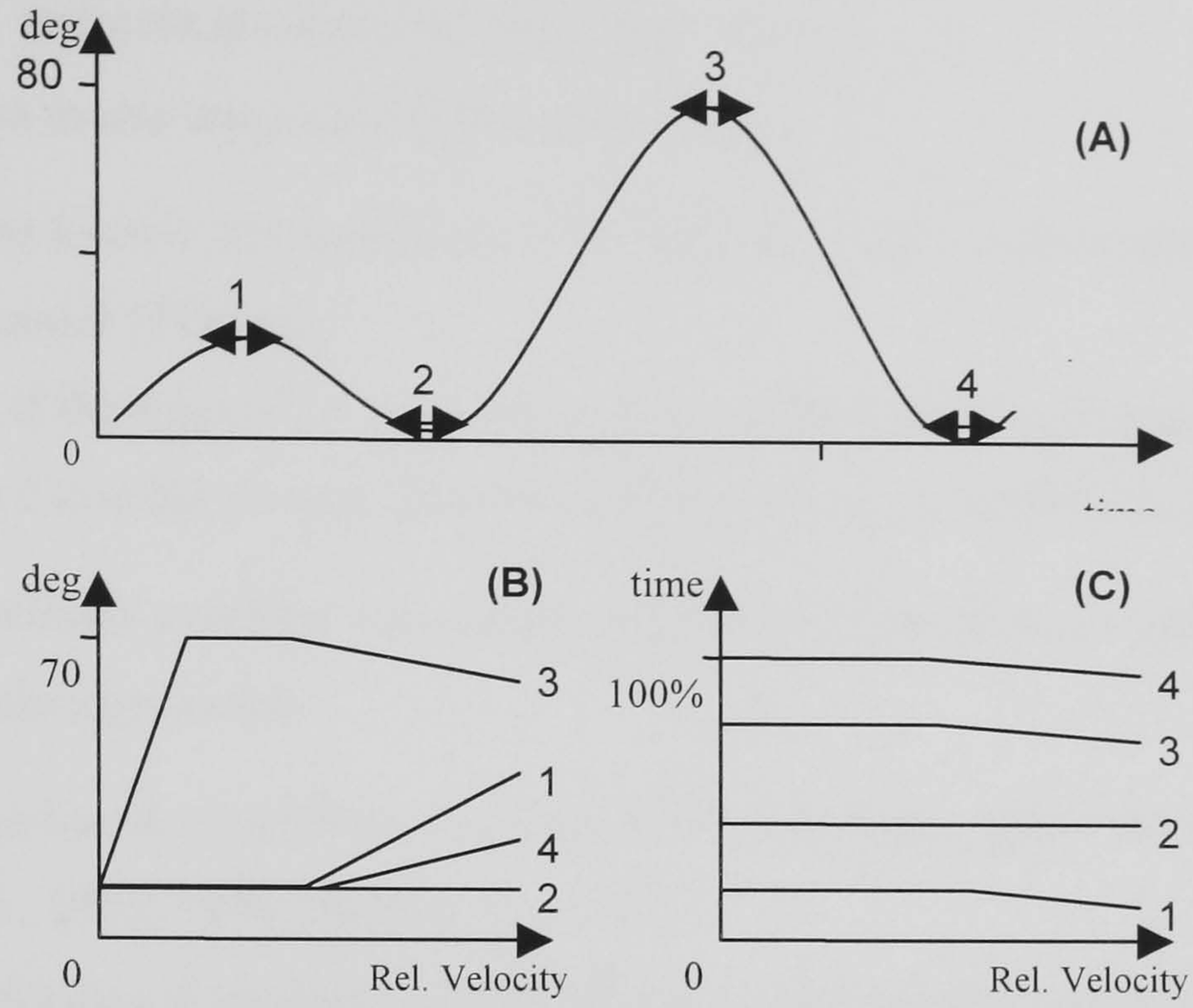


Figure 2.15: An example of parametric motion curves. a) knee flexion curve b), c) variations of the coordinates of its control points as functions of relative velocity

Dynamics-Based Methods

The methods introduced above are kinematics methods. Since these methods do not truly use physical laws to simulate the motion, they often produce physically incorrect motions. In contrast, dynamics-based algorithms use the laws of dynamics to synthesise the motion and thus they guarantee the physical correctness of the results.

Unlike kinematics methods, which only deal with time and the spatial characteristics of the motion, dynamics methods also consider the forces that cause the motion[1]. This affects not only the simulation algorithms but also the model; the specification of the model includes the dynamic properties of the object: mass, centre of mass, moment of inertia.

---

[1] There are also methods that use forces to solve secondary tasks like balance control [Boulic, 1997]. Though these methods take into account forces (force of gravity for instance) they cannot be considered as dynamics methods.

The physical simulation of the motion is based on Newton's laws:

$f = m \cdot \ddot{x}$

$t = i\ddot{\theta} + \theta \times i\dot{\theta}$, where f is the force, t is the torque, m is the mass, i is the inertia matrix, x and q are position and orientation vectors.

There are two ways to use these equations in simulation:

1. If the forces are known, the equations can be applied to calculate the motion. This is a *Forward Dynamics* (FD) task;
2. Alternatively, if the motion (or part of it) is known, the equations can be used to calculate the forces that cause the motion. This is a task of *Inverse Dynamics* (ID).

Most dynamics methods combine both Direct and Inverse Dynamics to calculate the motion. There are three main approaches:

- use knowledge-based controllers to define the forces and torques for the FD simulation [Hodgins et al., 1995, 1996; Faure et al., 1997];
- use a kinematics-based method to approximate the motion and assert its physical validity by reconstructing and correcting the applied forces [Ko et al., 1996];
- use global-optimisation techniques to find the motions that satisfy given dynamic and kinematics constraints (goals, key-postures) [Witkin et al., 1988].

The task of kinematics controllers is to define the angular trajectories that are used to animate the figure with Forward Kinematics. Similarly, the dynamics controllers estimate the forces and torques so that the motion can be calculated using Forward Dynamics. This estimation is usually done in two steps. In the first step, a kinematic knowledge-based controller (or a user) defines key-postures. As in kinematics-only methods, the controller uses finite-state machines controlled by high-level parameters. The difference between kinematics and dynamics approaches becomes apparent at the second step. While kinematics methods find the intermediate positions by interpolation between the key-postures (states of the machine), dynamics methods uses the keys to approximate the forces needed to move from one key-posture to the next one.

These forces are calculated using proportional-derivative controllers. The controllers define the behaviour of the joints by the following equation: $f = -k_p(q - q_n) - k_v (q' - q'_n)$, where k_p and k_v are positional and derivative gains, (q, q') is the current state of the joint and (q_n, q_n') is the next (desired) state of the joint.

Finally, the resulting motion is computed using a direct dynamics procedure that combines the calculated forces with collision forces, friction forces and balance constraints.

This method requires that a specific controller is designed for every simulated motion. The design of such controller is a difficult task, which includes a lot of manual tuning. Unfortunately, reusing a controller for another model presents a problem – the controller has to be tuned again.

There is a variation of this approach. At first, a kinematics procedure is used to create an initial estimate of the motion (the whole motion not just some key-postures). Then the Inverse Dynamics algorithm is used to reconstruct the forces and torques from this motion. After correcting the calculated forces and combining them with additional forces the direct dynamics algorithm is used to calculate the final motion. Thus, dynamics is used as a kind of a posteriori constraint that asserts the physical validity of a kinematics-derived motion.

Usually the inverse task is solved locally. This means that the algorithm does not use information about constraints at other instants of time. No doubt, this situation is quite different from the real one. Although we do not usually notice it, our brain plans our motion and performs corrections that are necessary to accomplish future goals or satisfy future constraints (for example we change the length of our step before approaching a staircase). To solve this problem, Witkin and Kaas [Witkin et al., 1988] proposed a method of constrained optimisation called *Spacetime Constraints*. The idea of the method is to look for the solution over the whole time interval, rather than subdividing the interval in small iterations and solving the task locally. This works as follows:

1. A user sets constraints (key-postures, footsteps, etc) on the final and intermediate states of the figure.
2. The program sets additional constraints on muscular forces, ground-contact forces and constraints that model physical laws.
3. A constrained optimisation technique (such as Sequential Quadratic Programming) is applied to compute unknown forces and positions, while minimising some cost function. For instance, the sum of squared muscular forces can be used as a cost function to minimise the energy spent in achieving the goals.

The method of Spacetime Constraints has several limitations. First, it is highly computationally expensive and it does not guarantee that the optimisation algorithm will converge to an acceptable solution. Second, the physical correctness of the motion does not necessarily mean that the motion will look natural.

Unfortunately, this problem is peculiar to all dynamics methods. Animation algorithms do not know true internal and reaction forces and they have to reconstruct these forces using an optimisation or inversion technique. Since the equations used are normally redundant, both

techniques have to rely on some criteria (cost functions) to choose the appropriate decision. However, these criteria deal with mathematical or physical aspects of motions and do not consider such vague aspects as realism or "naturalness" of motions. As the result, a motion may be optimal from the algorithm's point of view but may be unrealistic from the animator`s point of view. This is the main factor that limits wide use of dynamics methods.

2.2.4 Combining Different Approaches in Character Studio

The above is mainly a research view to the problem. It should also be interesting to consider the view of professional animators, since it is they who will finally judge which method is good or bad. This section reviews a commercial animation system, Character Studio, which is in fact used by professionals to create figure animation.

At present, Character Studio (CS) is one of a few commercial animation products that specialises in character animation. Though all principal animation systems (Maya, SoftImage, etc) provide tools for creating of this kind of animation, these tools are usually poorly integrated and limited by general-purpose functionalities like the support of linked hierarchies, import of Motion Capture data and interactive Inverse Kinematics. The developers of CS have advanced farther.

Character Studio is implemented as an add-in component (plug-in) for 3D Studio Max. It supports all the standard figure-animation tools as well as many advanced facilities. These include the automatic creation of skeleton models and a procedural animation tool for the creation of walking / running animations. The following is a brief summary of the principal CS features.

- *Automatic creation of skeleton models*

  As mentioned above, CS supports the rapid creation and parametric editing of skeleton models of biped characters. This feature is very convenient for animators because it greatly reduces the time spent on creating and setting up the model. However, it has disadvantages, too; the CS model is quite simple and the structure-changing restriction does not allow the use of more accurate models;

- *Motion Capture support*

  CS supports the import of MC data and allows MC-based animation to be combined with animation created by other means (keyframe, footstep algorithm). Also the plug-in implements a simple retargetting technique – if one scales a model the MC data (transitional part) will be adjusted to maintain the foot-ground contact;

- *Flexible-skin algorithm*

  Character Studio includes an advanced "skinning" algorithm, *Physique*. This algorithm allows a mesh to be attached to a CS' Biped object (or any other hierarchical object) and it will modify this mesh in accordance with the deformations of the object. Distinguishing features of this algorithm include Bulge Editor for modelling muscles, tendon-modelling tools, spline-based blending of the vertices for modelling complex deformation (such as the ones in spine), etc;

- *Footstep-driven animation*

  This is the most advanced and interesting feature of CS. It allows the process of animating the locomotion to be turned into manipulation of footsteps. The tool (Figure 2.16) procedurally creates the initial footstep sequences for walking, running and jumping; then it provides a method for editing the sequence. The algorithm has several interesting features: automatic balancing of the model, accurate processing of the foot during the contact phase and simulating different types of walking by allowing the heights and orientations of the footsteps to be changed. These features demonstrate that the footstep-driven algorithm is a mixture of different techniques: the algorithm is primarily IK-based; it uses dynamics to control body balance and uses biomechanics knowledge to create initial footsteps and to process the foot-ground contact. This confirms the fact that a practical animation algorithm should not be based on a single motion-control technique but should use several techniques.

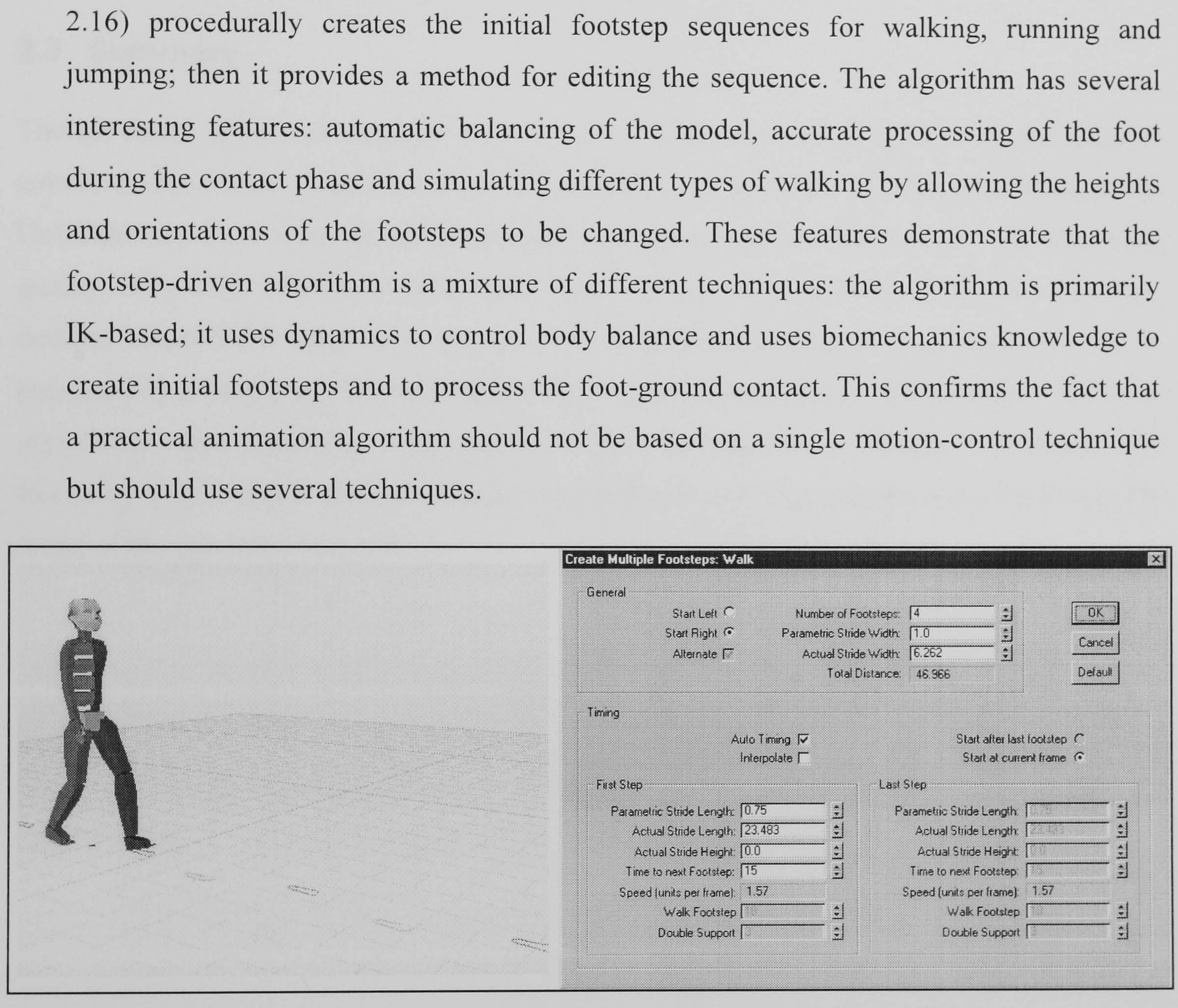


Figure 2.16: Creating footstep-based animation in Character Studio

All of these features make Character Studio a powerful and useful tool. However some drawbacks limit its use by exacting animators. The main problem is that CS is heavily based on the Inverse Kinematics algorithm, which does not take into account important features of

human gait[1]. Consequently, generated motions are not accurate from a biomechanical point of view. Another drawback is that the plug-in does not support constraints, which results in animation artefacts, such as feet penetrating the ground, or sliding on it.

Also CS has an intentional limitation: the plug-in is implemented as a closed system, which means that third party developers (researchers) or advanced animators cannot add new functionalities (new animation algorithms, etc) to it or introduce some modifications. Because of this limitation we could not use CS directly and had to implement our own "open" figure plug-in. However, CS was still extensively used at every stage of the research for analysis, testing and evaluation purposes.

2.3 Summary

Though many animation methods exist, none of them can satisfy all the requirements of animators. Procedural methods are convenient and can be used in a wide range of application. Unfortunately these methods still have many limitations and are incapable of creating high-quality animation. MC-based techniques can produce very realistic animations but they demand considerable physical resources and are time-consuming. A promising way out is presented by a family of Motion Retargetting / Warping techniques, which combine different approaches, thus benefiting from realism of Motion Capture technology, and power and flexibility of procedural methods. Exactly this approach was also adopted and extended in the scope of the presented research.

---

[1] CS is supposed to be used for creation of human characters as well as for any other bipeds. Therefore, there was no intention to create an accurate model of *human* locomotion.

3 Biomechanical Basis of Human Motion

The human body is a very sophisticated "mechanism" which behaves considerably differently from its mechanical analogies (robots). However the approaches that are used in Character Animation are mainly mechanics-based, adopted from robotics. As a result, the animations generated using these methods do not appear realistic; the generated motions look stilted and are more similar to the motion of a robot rather than that of a human. To improve the situation, animation algorithms use biomechanics knowledge about the gait, though this information is usually basic and is used for cosmetic corrections of the generated motion rather than being used in the simulation algorithm itself.

In this project, we have performed a deeper investigation of the biomechanics of human locomotion and used the knowledge gained as a basis for our models and algorithms. This chapter gives an overview of this basis: it discusses the principles of human locomotion, the behaviour of human joints and their role in human gait.

3.1 Structure and Functions of the Joints

It is no coincidence that the overview of the biomechanical basis of human locomotion starts with a description of the structure and functions of human joints. The joint is the corner stone of locomotion and we are convinced that proper modelling of the joints is important for the successful modelling of locomotion.

3.1.1 Joint Structure and Range of Motion

A joint is a bodily component used to connect the bones of the skeleton. It serves two main functions: mobility and stability. The structure and function of joints are closely related. The structure defines the character and ranges of motion available to the joint and thus it heavily influences the function of the joint.

The organisation of a synovial joint (the most frequent type of joint in the human) is not simple. On one hand, the joint should be stable and prevent the bones from disconnecting, but on the other hand, it should also allow them to move. This is achieved by means of a joint capsule, which encloses the ends of the bones and thus connects them, but not in a rigid manner. In order to obtain additional stability and strength for the joint, the bones are also connected by ligaments.

The character and range of motion in a joint is determined, to a considerable extent. by the cartilaginous surfaces, which cover the ends of the bones.

Other factors influencing the range and orientation of joint motion are the ligaments connecting the bones, and the joint capsule. Also, it is obvious that the muscles will have some effect on the range of joint motion.

3.1.2 Joint Classification.

Most joints in the human body permit some form of motion, though a small number do not. The former are divided into three groups – uniaxial, biaxial and multiaxial – on the basis of the motion that can take place.

A *uniaxial* joint has one degree of freedom, rotation about an axis, e.g. the interphalangeal joint of the fingers. A *biaxial* joint allows a motion about two axes, e.g. the knee joint. *Multiaxial* joints have three degrees of freedom, which can be rotations, as in the hip joint, or gliding, as in some spine joints.

Often, the motion of joints located close to each other are so interdependent that it is more convenient to consider them as a single unit rather then several joints. Examples of such aggregations, called joint complexes, are the foot complex, wrist, spine, etc. In the case of joint complexes, the term degrees of freedom is mainly used to refer to articulations of the whole aggregation rather than to movements available in single joints.

3.1.3 Instant Axis of Rotation.

Although it is often assumed that motion in a joint is a simple rotation about a static axis, this is not really true. The axis of rotation does not stay the same, but continually changes throughout the motion so that at any moment the rotation is being performed about an instant axis called the *IAR (Instant Axis of Rotation)* [Norkin, 1983]. Thus, it is necessary to have a set of axes of rotation in order to simulate the joint motion properly. However, in many cases the change between axes is negligible and a single axis is a reasonable simplification when simulating the motion – the error induced by this depends upon the joint and the type of motion.

An example of this effect is a flexion motion in the knee: the complex asymmetric structure of this joint and the position of the ligaments creates a situation where a simple rotation in the sagittal plane is transferred into a complex combination of rolling and sliding motions of one

bone on another. As a result, the position and the direction of the rotation axis is constantly changing.

3.1.4 Combined Motion.

When you perform a motion in a joint, you can see that apart from the primary motion (the motion that you control), some *secondary motions* occur simultaneously with the primary one. For instance, a flexion of the knee is accompanied by its rotation. Such secondary motion is also called *combined motion* [Van Sint Jan et al., 1998].

In contrast to the primary motion, you are not in control of the combined motion – it is controlled by the character of the joint surface and the tension of ligaments rather than by signals from the central motor cortex. Combined motion can also occur as a result of motion in several joints that form a single joint complex.

Mathematically speaking, the effect of combined motion occurs when the joint's axis of rotation is inclined in relation to the primary anatomical axes. And the larger the inclination, the more pronounced the effect of combined motion.

Note that so-called *associated* motion can occur in several joints simultaneously. This motion is automatic, too, but it has a different cause: tension of two-joint muscles (muscles acting across two joints).

In most cases, the amplitude of the secondary motion is much smaller than the amplitude of the primary motion. Nevertheless, it can be significant – for example, the amount of automatic rotation accompanying a full flexion of the knee is approximately 15°.

3.2 Major 'Locomotion' Joints

Walking is a difficult activity, with many parts of the body (joints, muscles, etc) participating in it. However some joints have particularly important roles in performing this locomotion. These are the joints of the lower limb as shown in Figure 3.1 (the hip complex, 1; the knee, 2; the ankle-foot complex, 3) and the spine. All of these joints (or complexes) are crucial for the simulation of locomotion and, therefore, the purpose of this section is to study them thoroughly prior to modelling.

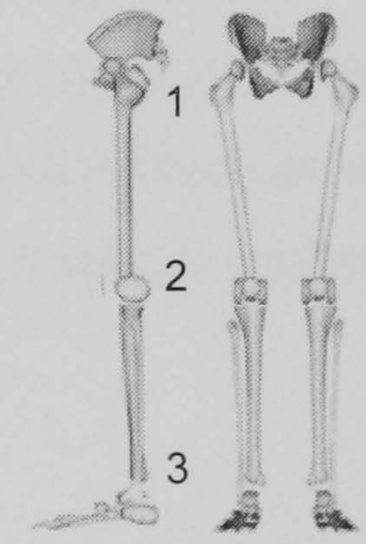


Figure 3.1: Lower limb

3.2.1 Hip Complex

The hip joint is a ball-and-socket joint (Figure 3.2) and has three degrees of rotation. It is not an ideal ball and socket, but it works almost like its mechanical counterpart, so its axes can be approximated as the axes of the orthogonal reference frame with a centre at the centre of the joint.

Figure 3.2: Ball of the hip joint

In contrast to the knee joint, all three degrees of freedom in the hip are important, even for straightforward walking. Together with the flexion/extension motions, the participation of which in walking is obvious, both the motion in the frontal plane, and the rotation should be considered in the simulation, or analysis. These are part of the *gait determinants*, which help to minimise the energy losses during walking by reducing the amount of vertical and transverse movement of the centre of gravity of the body. As far as figure animation is concerned, including these motions helps to make the animation more natural.

The motions of the hip are closely related to those in the pelvis. The three hip motions mentioned above are the motions of the femur on the pelvis. However there are also three counterpart motions of the pelvis that occur at the hip: anterior-posterior pelvis tilting, lateral pelvis tilting and pelvis rotation.

Modelling pelvis motions is rather complicated. First, these motions are the motions of a proximal bone, the pelvis, about the distal bones, the femurs, while the standard skeleton models are optimised to deal with movements of the distal bones. Second, the pelvis motions are always accompanied with movements of the femurs and the spine. Anterior-posterior tilting is coupled with spine flexion; lateral tilting, which occurs in the frontal plane, is accompanied by lateral flexion of the spine and abduction or adduction in the hip; and, finally, pelvis rotation is accompanied by rotations in the spine and the hips. The third problem arises from the first two: since there are two sets of motions acting on the hip it is easy to confuse them with each other.

3.2.2 Knee Joint

Though the knee is often referred to as a uniaxial hinge joint, it is quite different from its mechanical analogy. The complex asymmetric surfaces of the bones (Figure 3.3) and the system of ligaments produce motions in the joint, which are rather complicated. First, the knee is not a uniaxial but multiaxial joint. It has three degrees of freedom: flexion and extension; internal and external rotation; abduction and adduction (valgus). Second, these motions do not occur in anatomical planes and about static axes, which means that the effects of combined motion and instant axes of rotation take place.

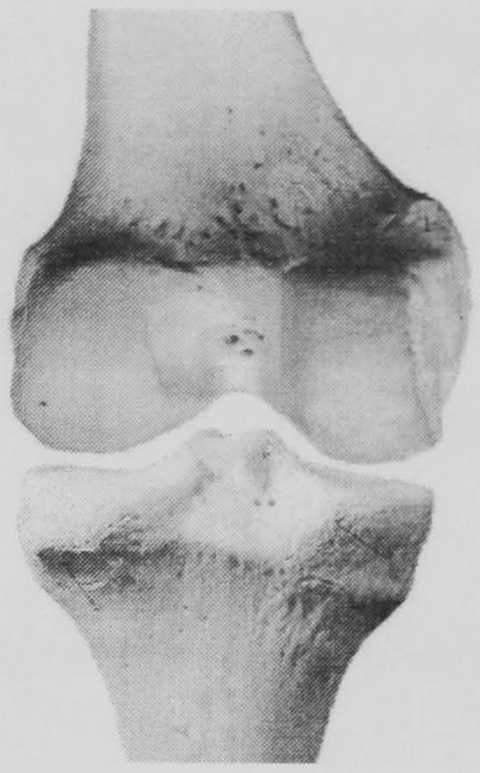


Figure 3.3: The knee

The main articulation of the knee is flexion. This motion mainly takes place in the sagittal plane, but there is also a combined rotational motion. The total amount of this rotation associated with full flexion is approximately 15°.

It is thus clear that the flexion motion of the knee is considerably different from that of a simple hinge moving in sagittal plane, which is its conventional representation in figure animation. Moreover, during the motion, the flexion axis of the knee not only changes in inclination, but it also changes in location. So it can be expected that using accurate axes for the knee motion will lead to a noticeable difference in the character of the animation.

The knee joint has two more DOFs: *rotation* and *abduction/adduction*. Rotation is performed about an anatomical axis and its range is dependent upon the knee position: when the knee is extended no rotation is possible, while the range reaches 70-80 degrees when the knee is flexed. In opposition to the combined rotation mentioned above this rotation is performed without combined flexion. There are few references to the role of this motion in the locomotion of the human. However, there is no doubt that the rotation takes place during the stance phase of walking.

As for the abduction and adduction motions, their ranges are very small and it does not seem justified to take these motions into account for the purposes of modelling locomotion.

3.2.3 Ankle and Foot

The ankle-foot complex is a structure that unites several joints of the lower part of the leg. Because the structure of the foot is quite complex (it contains about 30 bones), the motions within this formation are closely dependent. As a result, the character of foot motion is very complicated. Combined motion takes place in all joints of the foot. Moreover, sometimes the magnitude of the combined motion is so large that it is hard to tell the combined motion from the primary one.

Although there are many articulations in the complex, we limit this overview (and the model) to the three main articulations. These are:

- Plantarflexion/dorsiflexion in the ankle joint;
- Inversion/eversion in the subtalar joint;
- Flexion/extension of the metatarsophalangeal joint.

The drawing on Figure 3.4 illustrates the difference between the mechanical and anatomical views of the foot. The solid lines correspond to the real axes of rotation and the dashed lines to the simplified orthogonal axes. The drawing shows that the difference between the real and simplified axes is large (the inclination of the axis of the subtalar joint (2) is about 42°). Thus, it is likely that the real motion and the motion simulated by means of the simplified model will be quite different. Moreover, both articulations have a reasonable range of motion, which means that their combination can produce a wide range of motions not limited to rotations about just one of the foot axes.

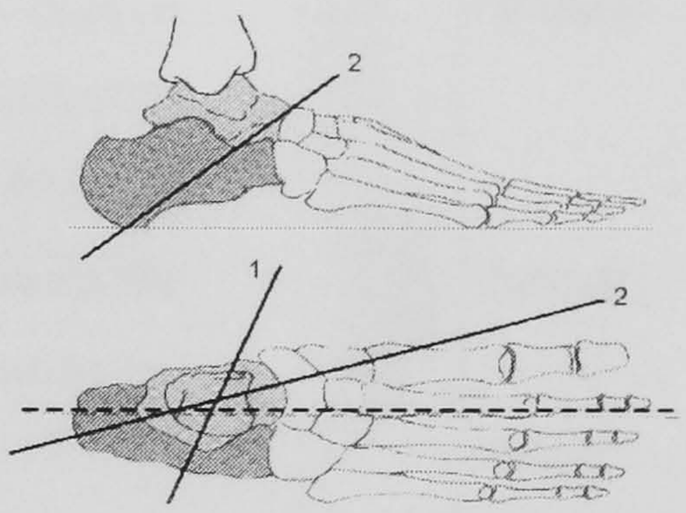


Figure 3.4: Rotation axes of ankle (1) and subtalar joint (2)

While the ankle joint is included in almost all figure models, the subtalar joint and the metatarsophalangeal joint rarely are. However, they play an important role in human locomotion, so it is worthwhile considering them, too. The biomechanical functions of the joints in the ankle and foot are to provide additional degrees of freedom, absorb the shock of weight bearing and smooth the motion. However, if you look at the animation of many computer-generated characters, you will see that they lack these features - they walk like skiers in heavy, rigid ski boots. Though there are several possible causes of this problem, one of them is an inadequate foot model. And the proper simulation of the main articulations of the foot and the accurate use of pivot points (heel, metatarsal heads, and toe) can make the motion of the character smoother and more natural.

Note: A shoe is the factor that can seriously affect the motion of the foot. In this case (and this is virtually always the case) accurate specification of the axes becomes less important than the correct positioning of the pivots and specification of the ranges of motion (or shoe flexibility).

3.2.4 Pelvis and Spine

The role of the spine in walking is far less important than the roles of the lower limb joints. However the motion of the spine and the upper body in general plays a significant part in the perception of the locomotion. Our observations suggest that a person is more likely to notice artefacts in the motion of the body than in the motion of the limbs. And, therefore, if one wants to produce a realistic animation of locomotion, one should consider using an anatomically accurate model of the spine.

The spine is the most complex part of the skeleton. It consists of thirty-three bones (vertebrae) that are stacked one on the other. For simplicity the vertebral column is usually divided into five regions (Figure 3.5); the vertebrae within a region have a common structure and similar articulation characteristics. The vertebrae are distributed in the following way: seven in the cervical region (neck), twelve in the thoracic region, five in the lumbar and sacral regions and four in the coccygeal region.

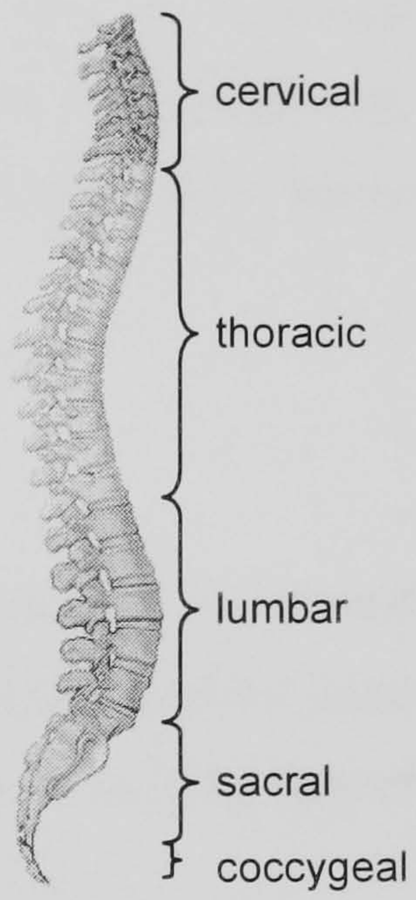


Figure 3.5: Spine regions

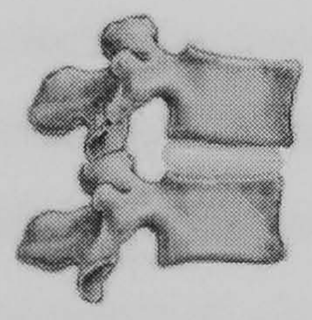


Figure 3.6: Vertebrae

A typical vertebra (Figure 3.6) (from the cervical, thoracic or lumbar regions) consists of two main parts: the anterior part, called vertebral body, and the posterior part called the vertebral arch. Articulations of these parts with corresponding parts of adjacent vertebrae form two joints: the intervertebral joint composed of two vertebral bodies and an intervertebral disk interposed between them; the facet joint formed by articulating facets of the arch. The motions available in these joints are quite different. The facet joint is a plane joint so its primary motion is gliding. The intervertebral joints allow more motions: rotating, gliding and tilting (this motion is possible due to flexibility of intervertebral disks). The motions in both joints are interdependent – a combination of gliding in the facet joint with tilting, rotating and gliding in the intervertebral joint result in three composite motions: flexion, lateral flexion and rotation.

The amount of motion between two vertebrae is very limited, but the compound effect of the motion in many vertebrae makes a significant range of motion. Since there are strong

dependencies between motions in individual spine joints, it is possible to define compound motions of the spine. The contribution of each intervertebral joint on the compound motion can be defined as proportion of the total motion (see Appendix). For instance, the rotation in the neck is distributed between the joints of the cervical region as follows: 50% at the first joint (atlantoaxial), 5% at the second joint, about 10% for each of the next four joints and 5% at the last joint. Such an approach significantly simplifies the modelling of spine motion as it allows a decrease in the number of DOFs. However, one cannot limit the model to just three motion of the spine (flexion, lateral flexion and axial rotation). As a minimum requirement, it is necessary to consider the motion in the neck and the rest of the spine separately.

Another interesting aspect of the motion of the vertebral column is that the basic motions are often interdependent (an effect of combined motion). For instance, due to structural characteristics of the facet joint, the lateral flexion in the neck is always accompanied by rotation. However, this effect has little importance on locomotion and, therefore, there are no reasons to incorporate it into the model.

3.3 Joint Measurements

Obviously, if one wants to create an anatomically accurate model of joint motion, one should base it on real data. Usually biomechanical engineers are interested in two aspects of the joint: its morphology and its motion. For the purposes of animation, it is sufficient to have data of joint motion only.

One of the most accurate and straightforward approaches to measuring motion is to use a 3D-electrogoniometer [Van Sint Jan et al., 1999], which takes measurements as follows. The electrogoniometer is fixed to the subject's body with one end of the goniometer attached distally and one proximally to the measured joint. Any motion in the joint changes the strength of variable resistors that form the goniometer and a computer connected to the device can track the motion by taking the readings and converting them to angle or distance values. The resulting motion data are presented in the form of a series of transformations that correspond to the 3D rotations and translations of one bone relative to the other. Knowing this transformation, one can easily derive the parameters needed for the skeleton model (i.e. the position and orientation of the axes and limits of the motion).

Since the aim of the project was to learn about the utility of anatomical modelling for animation, rather than to achieve maximum accuracy in figure modelling, we found it appropriate to rely on averaged data that were available in the biomechanics literature.

As the later experiments demonstrated, this was the right decision: errors introduced by other data, inconsistency between the original skeleton and the model and, most importantly, simulation error would make the use of more accurate data unjustified.

3.4 Functional Walking Model

Before proceeding with the overview of human walking, one should try to understand why we are so interested in this particular kind of human activity. What are the motives of many researchers, including authors of this work, to enter this field?

There are several reasons for this. First of all, walking is on of the most common human activities and therefore there is a great need for walking animations. Also, the fact that this activity is so common, and we are so familiar with human gait, raises the requirements on the animation and consequently more and more advanced animation algorithms are required. Second, it would be no mistake to assert that joints of the lower body are particularly adapted for performing their roles in this primary kind of locomotion. Therefore, simulation of walking is the task that would especially benefit from using an advanced approach to joint modelling.

3.4.1 Definition of Walking and Gait

Because walking is such a familiar activity, there is no need to define it here. The only question that can arise about its definition is how to distinguish walking from running. The criterion here is the constant presence of foot-ground contact: in walking at least one foot should be in contact with the ground at all times.

The term *walking* unifies many different kinds of walking-type locomotion; there are an infinite number of different gaits, there is walking with or without turning, walking on flat or inclined surfaces, walking up/down a staircase, and many more. Obviously, it is impossible to give a description of gait that is both detailed and covers all different kinds of walking. Therefore the overview given in this chapter describes *the normal or standard gait*. To some extent, this description can be generalised for many other gaits that do not come under the definition of the normal gait.

This also means that some algorithms based on this information are intended (and tested) for use with standard walking motions. To apply these algorithms to other "walking" locomotion (walking up/downstairs, walking on a slope, running) they should be correspondingly revised.

3.4.2 Phases of the Gait

Since walking is a rhythmic activity, it is possible to divide it into repeating periods or cycles. In principle, any repeating gait event can be used to divide the motion into these cycles, though the most accepted variant, which is used in biomechanics literature, is to use the initial foot-ground contact as a starting point of a cycle. Thus the *gait cycle* is defined a period of time between two initial foot-ground contacts of the same extremity.

Then, during a cycle each extremity passes through two distinctive phases: the *stance phase*, when the foot is in contact with the ground, and the *swing phase*, when the extremity is unsupported.

The *stance phase* starts at the moment the foot contacts the ground (*initial contact*) and ends when the toe of the same foot leaves the ground (*toe-off*). Through the whole stance phase the foot is on the ground, though at different moments different parts of the foot are in contact with the ground: heel, flat foot, forefoot and toe. The stance phase constitutes about 60% of the gait cycle.

The swing phase begins after the toe leaves the ground and finishes just before the foot hits the ground again. In normal gait, the swing foot never touches the ground. The swing phase is always shorter then the support phase and makes up about 40% of the cycle.

In gait, the stance phases of left and right legs overlap creating a period of *double support*. The presence of double support is the criterion that distinguishes walking from running. Since there are two stance phases in one cycle, there are also two periods of double support, which occur approximately from 0 to 10% and from 50 to 60% of the gait cycle.

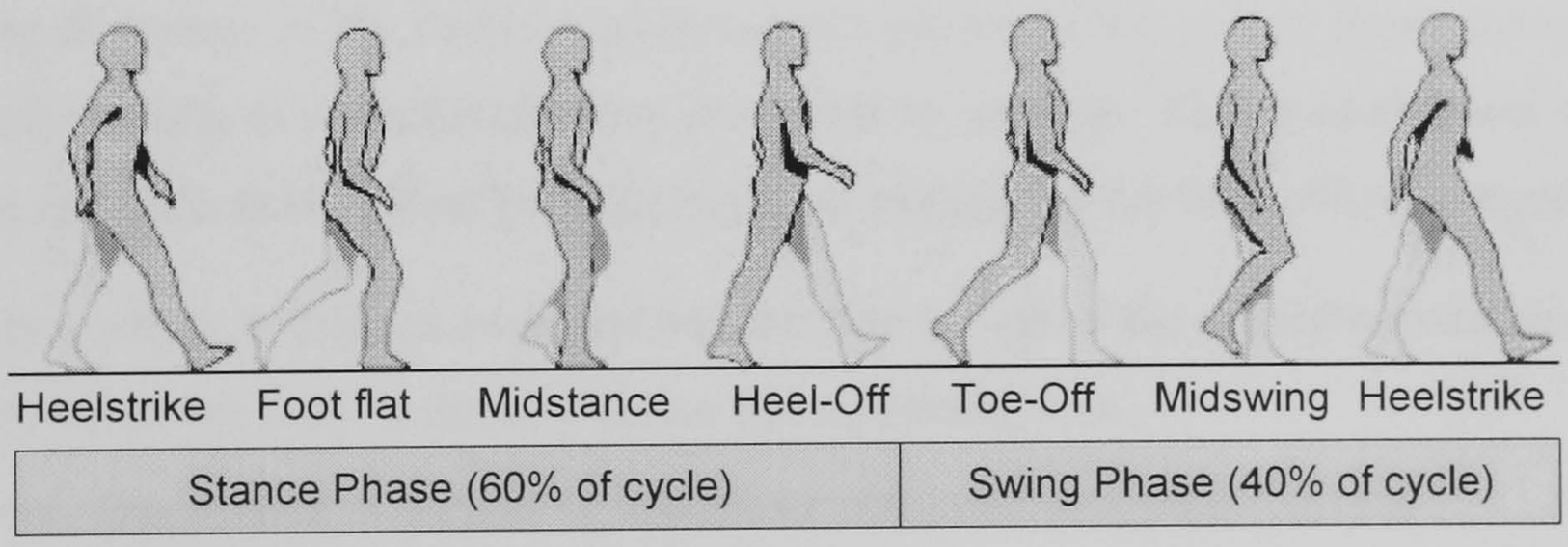


Figure 3.7: Main events of the gait cycle

A description of walking can be significantly simplified if one uses some characteristic events to refer to particular periods of the gait cycle. Here are the most important of these events (Figure 3.7):

- *Initial contact (or foot strike)* marks the beginning of the stance phase. At this moment the foot of the leading limb makes contact with the ground. In normal gait this contact is made with the heel (*heelstrike*), while in pathological gait or in non-standard gait the contact can be made with the flat foot or the toe. The ground-reaction force arised during the initial contact can also vary significantly among different individuals and among different gaits;
- *Foot flat* refers to the instant of time when the foot becomes plantar grade on the ground. This event usually occurs at 8% of the gait cycle;
- *Midstance* is an instant of time when the swing limb passes the stance one side by side. This event occurs approximately in the middle of the stance phase about 30% of the whole gait cycle;
- *Heel-off* is the point in the stance phase when the heel leaves the ground;
- *Toe-off* event finishes the stance phase. At this moment (about 60% of the cycle) the toe leave the ground and the swing phase begins;
- *Midswing* of the swinging limb coincides with *midstance* of the stance limb; at this moment both feet are positioned side by side.

Subdivision of the stance and swing phases

The decomposition of the gait cycle into the stance and swing phases is too rough to be effectively used in gait analysis. Therefore, these phases are usually divided into sub-phases. Unfortunately, there is no single fixed definition of these sub-phases: various authors use different decompositions or imply different meanings to the same terms. The decomposition used here was introduced at the Gait Laboratory, Rancho Los Amigos in California, and is now the most widely accepted gait decomposition in gait analysis now.

- *Loading Response* is the initial double-support period of the stance phase during which the body weight is transferred from one limb to another. This period starts at initial contact and ends at *toe-off* of the opposite limb, occupying the first 10% of the gait cycle;
- *Midstance phase* is defined as period between the *toe-off* of the opposite extremity and the moment when the body is directly above the supporting foot;
- *Terminal stance* starts at the end of *midstance* and continues until *heel-off*;
- *Preswing* phase is the finish of the stance phase. It starts with *heel-off* (or *heel rise*) and ends when the toe leaves the ground at *toe-off*, constituting about 20% of the gait cycle;
- *Initial swing* corresponds to the first one third of the swing phase from *toe-off* to the point where the maximum knee flexion occurs. Alternatively, the swing phase is often

decomposed into initial swing, midswing and terminal swing phases, each constituting about one third of the whole swing phase;

- *Midswing* period starts at the point of maximum knee flexion and continues until the tibia becomes vertical. This period also occupies about one third of the swing phase;
- *Terminal swing* is the final part of the swing phase during which the knee extends in preparation for the initial contact.

Time and distance parameters of the gait

The following parameters are used to describe the time/distance characteristics of walking: *Stride length* is defined as the distance between two successive ground-contact points of the same foot. In normal gait, where the left and right steps are identical, this value is equal to double the step length. The stride length value depends upon several factors including the height of the person, age, type of the gait and walking speed.

Cadence measures the number of steps taken per unit of time (minute). The cadence varies with the speed of walking: it increases in fast walking and decreases in slow walking. Natural (or free) cadence refers to the normal walking rhythm inherent in every subject. This rhythm varies with age: it is about 180 in childhood, but the average cadence gradually reaches a value of 120 steps per minute by 18 years, after which it changes only slightly.

Speed measures the distance traversed by the body within a unit of time. The instantaneous speed, which is defined as a derivative of the body's path, changes from moment to moment. Therefore, it is more convenient to use an average speed, which can be calculated by multiplying cadence and stride length: *speed = (stride length) x (cadence) / 120*

3.4.3 Gait Determinants

One of the main conditions of efficiency of walking is minimisation of the excursion of the body's centre of gravity. There are six main factors that affect the efficiency and energy expenditure of walking. These factors are called *gait determinants* [Whittle, 1991].

The first five determinants (lateral pelvis tilt, knee flexion in stance phase, pelvis rotation, ankle and foot motion) are responsible for reducing the vertical movement of the centre of gravity (*COG*) and smoothing its path (Figure 3.8). The sixth determinant (lateral pelvis displacement) is concerned with narrowing the side-to-side movement of the body. Here

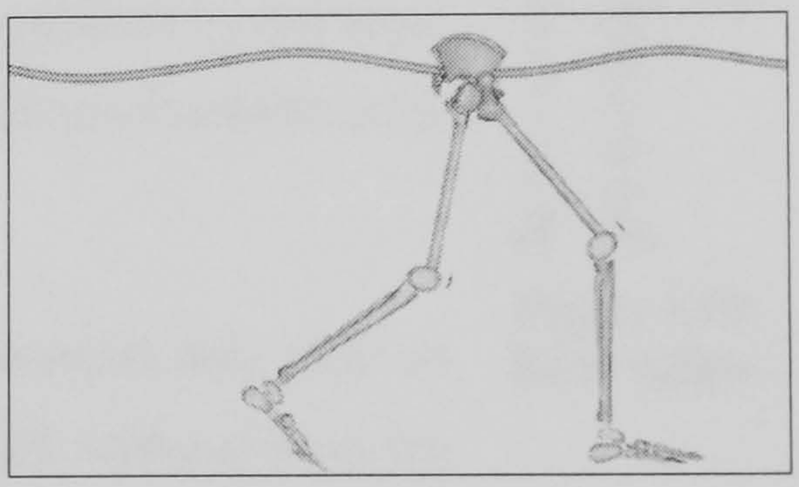


Figure 3.8: Sinusoidal path of COG

are the detailed descriptions of the determinants:

1. *Lateral pelvis tilt* or *pelvis drop*: Following the *initial contact* the body's COG starts to rise and reaches its highest point at about midstance. To reduce this rise – and in that way to minimise the vertical movement of the COG – pelvis drops on the side of the unsupported extremity (Figure 3.9). This motion is accompanied by abduction and adduction of the hips and by lateral flexion of the spine to the opposite side;

Figure 3.9: Pelvis drop

2. *Knee flexion in stance phase*: Flexion of the stance knee is another adjustment that does not allow the body to rise too much in midstance;

3-4. *Interaction of knee, ankle and foot*: The smoothness of the body's COG pathway is also an important criterion of gait efficiency. This smoothness is achieved by combined movements of the knee, the ankle and the foot, which either shorten or lengthen the extremity to prevent any abrupt changes in the vertical position of the COG. For instance, following the initial contact the body would rise abruptly if flexion in the knee and plantarflexion in the ankle did not shorten the limb making the transition more gradual;

5. *Pelvis rotation*: During gait, the forward and backward motions of extremities are accompanied by pelvis rotations in the transverse plane. When the swinging extremity moves forward, the pelvis also rotates forward about the hip joint of the supporting limb. This effectively lengthens both stance and swing extremities, which in turn helps to decrease the drop of body's COG. Naturally, the pelvis rotation has to be counterbalanced by compensatory rotations of the hips and the spine. The total amount of pelvis rotation varies significantly among gaits and people, and its average value can be approximated as 8 degrees for men and 10 degrees for woman;

6. *Lateral displacement of body:* This determinant is responsible for narrowing the width of the base of support and thus reducing the amount of side-to-side motion needed to transfer body weight from one extremity to another. This becomes possible due to a slight angulation in the knee (*physiologic valgus of the knee, Figure 3.10*), which allows the tibia to be vertical when the hip is adducted.

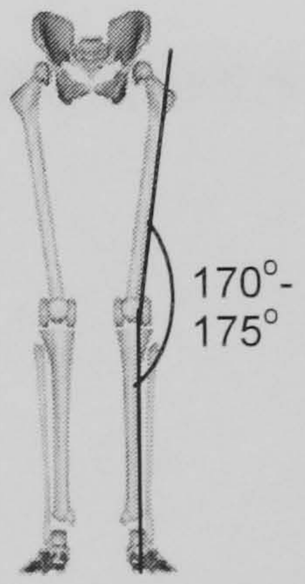


Figure 3.10: Knee valgus

Though the gait determinants may seem small and inessential they play an important role in the way we perceive the gait. First of all, without them the

gait would not be so smooth. Secondly, the values of determinants vary among persons and different kinds of gait, therefore they provide us with important information about the individual characteristics of a particular gait.

3.5 Roles of Joints in Gait

The preceding sections have introduced the basic structure and functions of human joints and given an overview of walking, the main type of human locomotion. This knowledge forms the basis for the analysis of human joints in locomotion (walking), which is given below. The analysis concentrates on the following topics:

- *The roles of particular joints or joint motions in gait:*
  This knowledge helps us to decide whether a particular joint motion should be included into the walking model, and if so, how we can modify the motion while preserving its main features and character. To answer these questions, a description of kinematic and dynamic behaviour of joints in walking is presented;
- *Correlations between different joint motions and between motions and phases of the gait:*
  Again, knowing these correlations helps us to find a way to adjust a motion without disrupting its integrity. For instance, if it is known that some feature is inherent in walking and has an invariable position in the gait cycle then we should not apply any modification to a motion that can remove this feature or change its relative position in the gait cycle. Additionally, it helps to improve accuracy and reliability of automatic gait analysis, which forms the kernel of our animation algorithm;
- *Features of joint-motion curves*: Here the motion is analysed from a mathematical point of view. Mathematical analysis of joint curves allows important features of the motion (special points, smoothness and shape of curves' segments, range of motion) to be picked out and their characteristics to be specified.

As one can see, this analysis concentrates on answering practical questions, suggesting how to design a biomechanically accurate MC-based animation algorithm.

3.5.1 Hip Joint

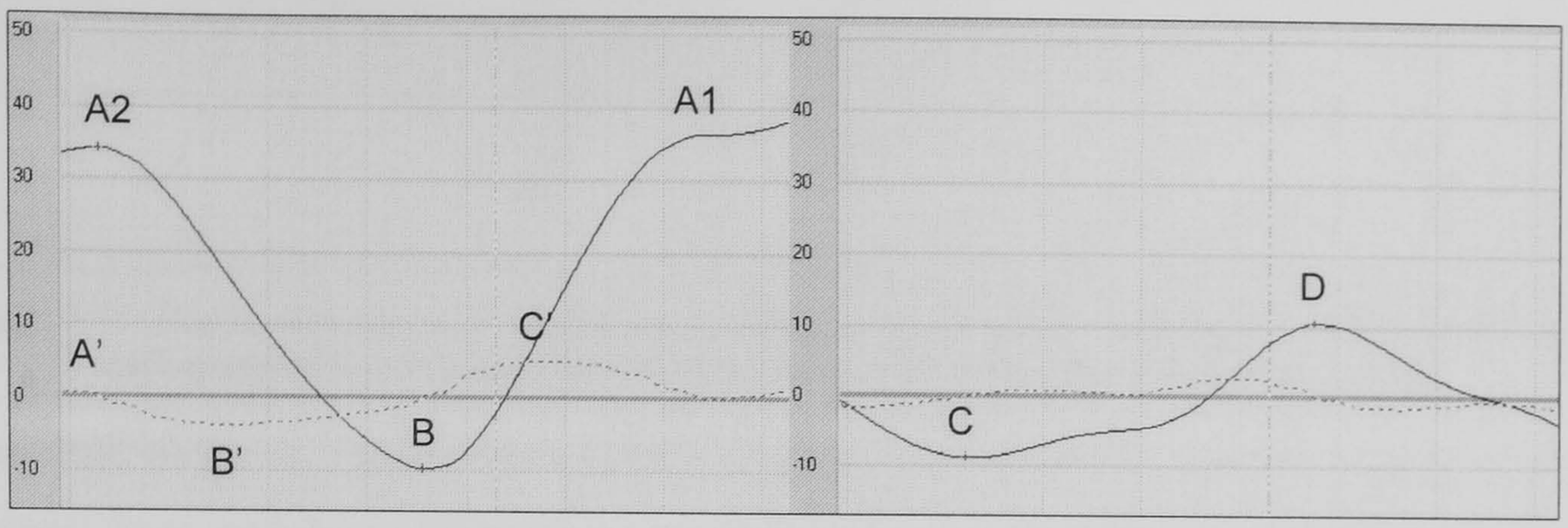


Figure 3.11: Joint angle (solid) and speed (dotted) for hip flexion (left) and abduction (right). Note for joint-motion graphs in this chapter: vertical scale is in degrees, horizontal (time) scale is in frames, vertical dash-dotted line splits the stance and swing phase.

Role of the hip flexion in the gait

Stance phase: The joint enters the stance phase at about 35 degrees of flexion and then steadily extends through most of the phase. During the first half of the stance phase the external momentum created by the ground reaction force opposes the extension (and can even create a second flexion peak, A2 on Figure 3.11) and therefore significant muscle activity is needed to extend the hip. Around midstance, the roles of the external and internal (muscle) moments are switched and the muscles now works to flex the hip. Between *heel-off* and *toe-off* the hip reaches its maximum extension (B) of about 10 degrees and starts to flex again. The flexion moment also continues to increase until *toe-off* (C').

Swing phase: The hip flexes during almost the whole swing phase. After reaching the flexion maximum just after midswing, the hip position does not change much either extending or flexing a bit. This serves a function of increasing step length and reducing the impact when the heel hits the ground.

Role of the hip abduction and rotation in the gait

Basically, these motions mirror the corresponded pelvis motions in order to maintain the swinging leg (and the whole body) closer to the optimal progression line. These motions will be reviewed in the pelvis section (Section 3.5.4).

Curve analysis

The hip flexion curve has a distinctive pattern consisting of alternating flexion and extension peaks. The flexion peak takes place before or just after the initial impact and it is generally not as steep (Figure 3.12a) as the extension peak or it can be divided into two sub-peaks (Figure 3.12b). The maximum extension is during the *preswing* period, just before *toe-off*.

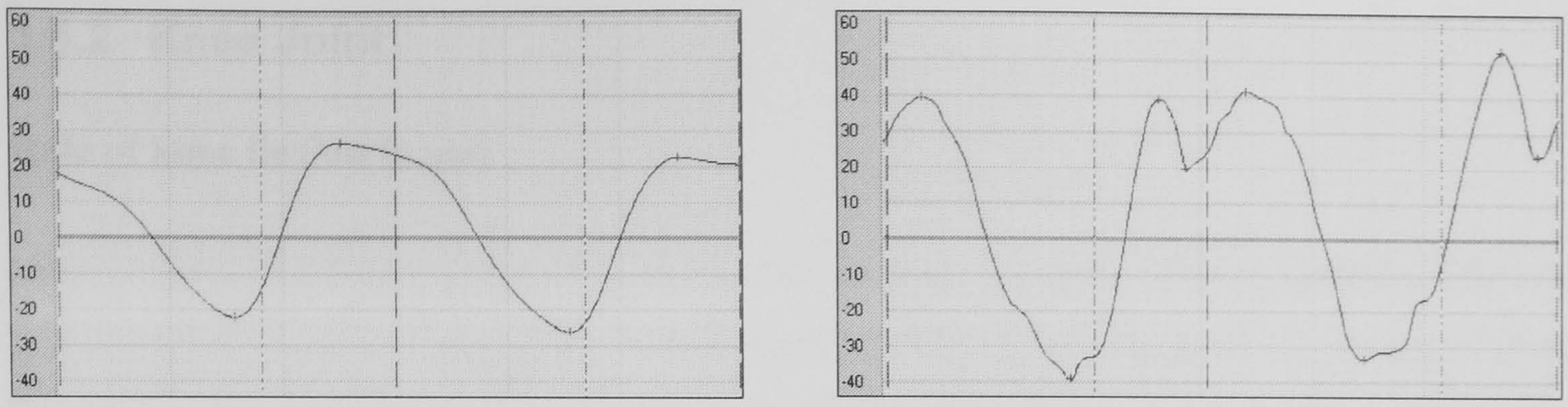


Figure 3.12: Hip flexion curves are different for different gaits (a. fast walking, b. waddling walking); each graph spans two gait cycles to show variability of the motions between cycles.

The normal range of hip flexion is about 20 degrees of extension and 30 degrees of flexion. However these values correspond to normal gait only and can vary significantly between subjects and different types of walking. For example, the amount of flexion used by an elderly person during walking may not exceed the interval of 5 to 35 degrees of flexion. This diversity can complicate any feature-recognition algorithm since it is almost impossible to use any absolute values for analysis purposes. The way out of this problem is to use relative values though this approach has some drawbacks, too.

It should be also noted that the flexion angle is measured between the pelvis and the thigh and therefore its range is dependent upon the range of pelvis flexion. Alternatively one can deal with the angle between the thigh and the vertical; this angle shows less variability and consequently it is often used in biomechanics and animation instead of the true flexion angle.

Synopsis

- All hip articulations are important for walking; the rotation and abduction motions of the hip are replicas of the corresponding pelvis motions, so their characteristics will be discussed in the pelvis section.
- The hip cycle can be divided into two distinct intervals: extension and flexion. In both intervals, the sagittal-plane motion of the hip is monotonic and smooth. The exception is the beginning of the extension interval: following the point of maximum flexion, the hip stays in a relatively constant position, flexing or extending a little.
- It is difficult to pinpoint positions of the intervals precisely: the maximum hip flexion, which marks the beginning of the extension interval, is located at about 85% (+/- 5%) of the gait cycle; the maximum extension occurs at the beginning of preswing, at about 50% of the cycle.
- In normal gait, the flexion curve has a point of inflexion (C') at the time of *toe-off*.
- The range of motion varies in wide limits with an average interval of 20 degrees of flexion and 30 degrees of extension. Sometimes it is more convenient to use an angle between the thigh and the vertical instead of the true flexion angle since the first one shows less variability.

3.5.2 Knee Joint

Role of knee flexion in gait

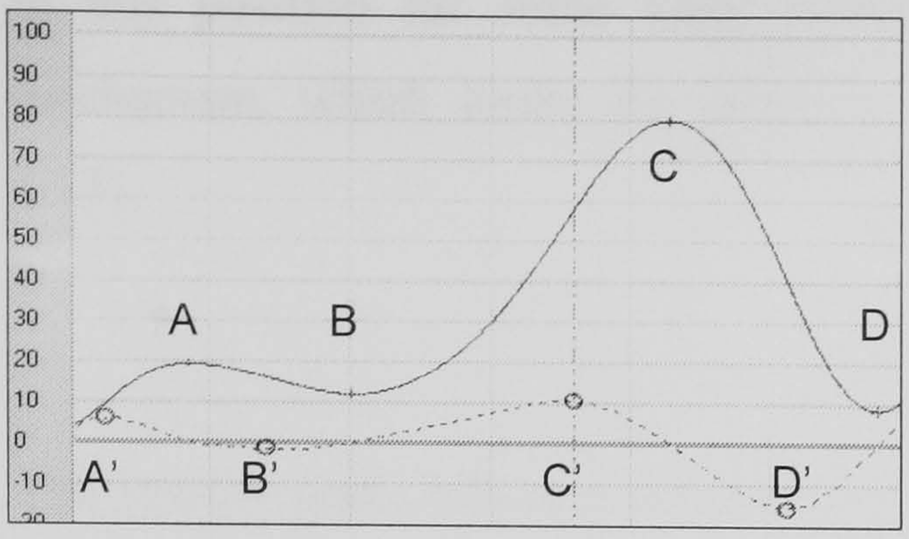


Figure 3.13: Knee flexion curves: angle (solid) and velocity (dashed)

Stance phase: During loading response the knee rapidly flexes from almost full extension to about 15-20 degrees of flexion (A on Figure 3.13). This initial flexion serves two purposes: impact absorption and minimisation of vertical translation of the body (third gait determinant). The velocity graph shows two different moments acting on the knee during this phase: an external flexion moment produced by body weight and a counteracting extension moment produced by contraction of the muscles.

As the tibia advances further, the ground reaction force acts in front of the knee, producing an extension moment. Again limb muscles come into play (B') to prevent hyperextension of the knee. The extension peak (B) is reached about *heel-off* time, after which the knee is steadily flexing in preparation for the swing phase. And by the time of *toe-off* the knee is usually flexed to 40/50 degrees. The resulting rotating moment acting on the knee changes rapidly (C') at *toe-off* though the knee will continue to flex just until the *midswing*.

Swing phase: The motion of the knee during the swing phase is mainly defined by the flexion of the proximal hip joint. Thus the whole limb works as a two-link pendulum and therefore no muscular activity is needed to move the knee. Only at the end of swing phase the knee muscles come into play again preventing the knee from hyperextension.

Curve analysis

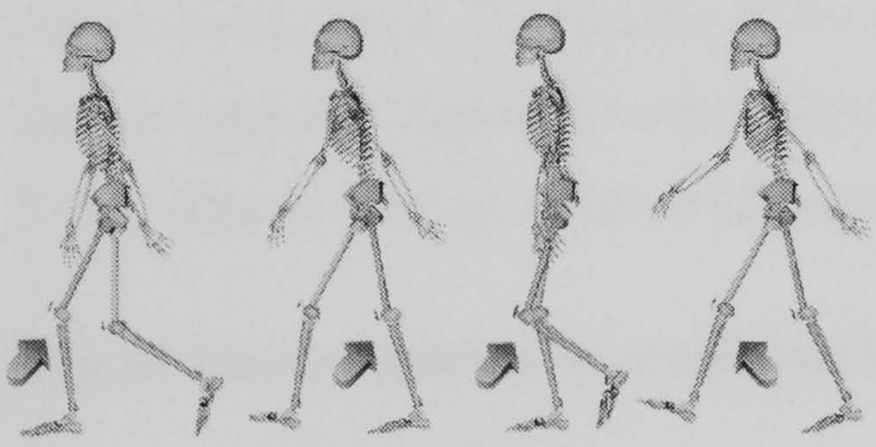


Figure 3.14: Postures corresponded to the knee-flexion curve keypoints A, B, C, D from Figure 3.13

As the graph (Figure 3.13) shows the knee-flexion curve has a prominent pattern: it consists of two peaks: a small one and a larger one. The gait cycle starts at beginning of the small peak, after the knee has straightened at **D** (Figure 3.14) and started flexing again. The joint continues to flex until the first local maximum (**A**) is reached between *flat foot* and *midstance,* and then it starts extending again. After reaching its second minimum (**D**) about *heel-off* time the knee steadily flexes up to the highest peak (**C**).

The knee does not necessarily reach full extension at **D**: depending on the gait, the maximum extension angle varies between 0 and 15 degrees. The peak can also be less pronounced: the extended knee tends to stay in this position for some time, probably due to the knee locking mechanism, which limits the mobility of the extended knee (Figure 3.15).

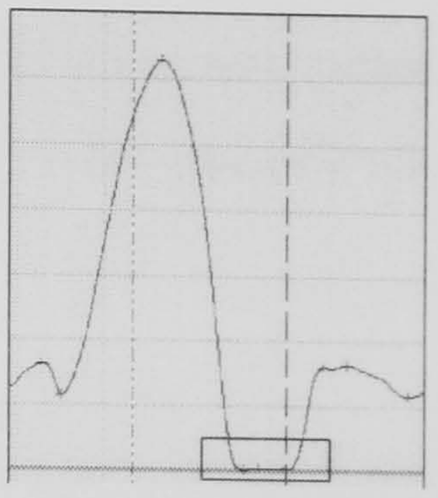
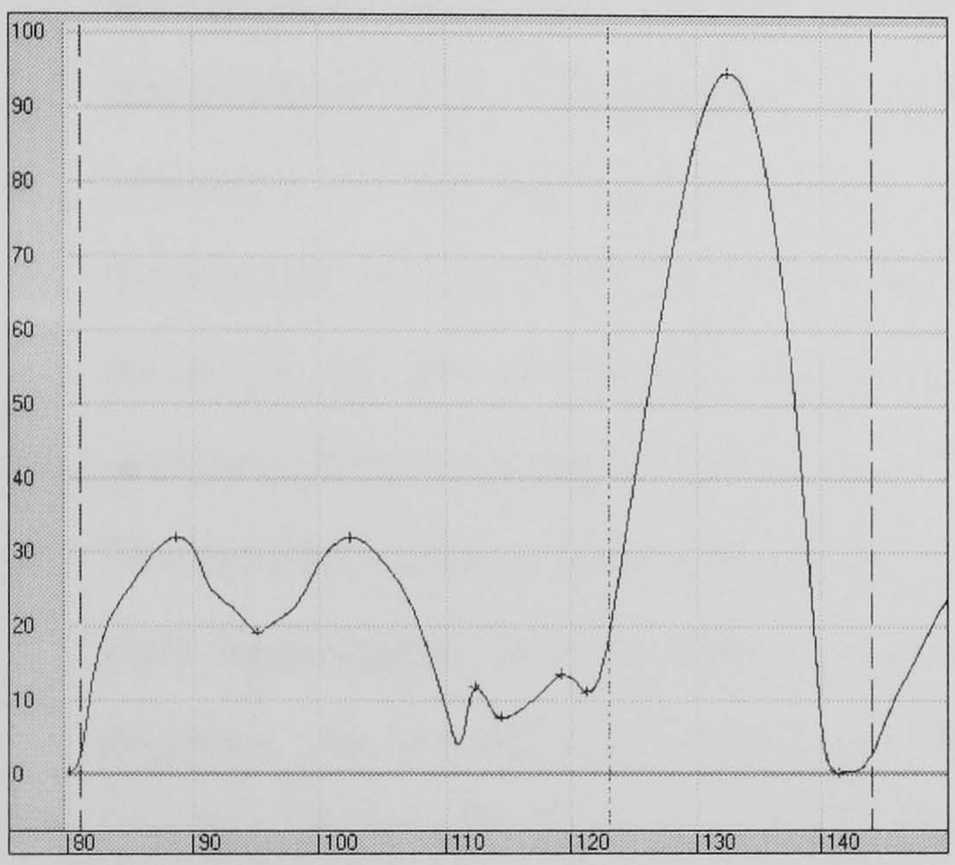


Figure 3.15: Knee extension peak

The smaller peak, which is defined by points A and B, is not as characteristic for the general gait as the larger one is. For some gaits, more than one peak is typical (Figure 3.16a) while some may not exhibit this peak at all (Figure 3.16b).



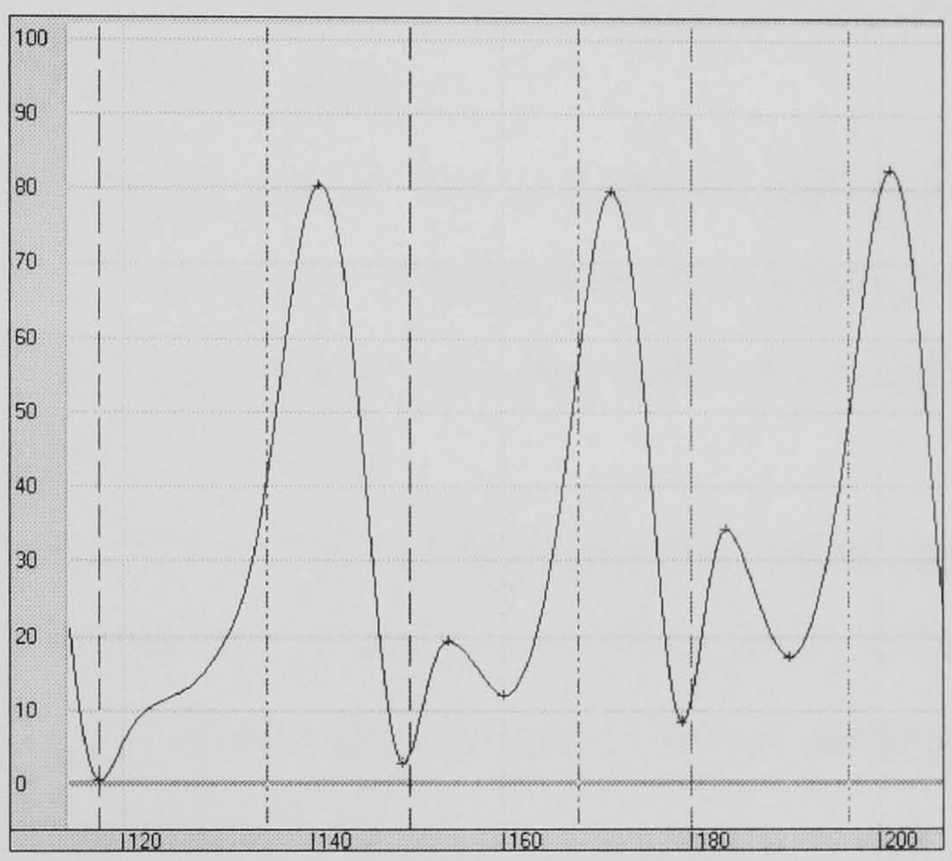


Figure 3.16: a) Exaggerated walking: the small peak is split into two parts
b) Walking with turning (the first cycle): the small peak is degenerated

While the shape and width of the small peak may vary greatly between gaits (and even within the same gait - Figure 3.16b), the shape of the main peak (**C**) is the same for all but the most pathological gaits, which can be explained by the pendulum nature of lower-leg motion during swing phase. Uniformity of the shape of the peaks is compensated by the diversity of their heights: the maximum value of knee flexion fluctuates within wide limits (Figure 3.16a, 3.16b) with an average value of 65 degrees.

Role of knee rotation and abduction in gait

As was said earlier, the common assumption that the knee is a one DOF joint is not correct. Besides combined rotation, which takes places due to obliquity of the flexion axis, the range of knee motions includes limited amounts of inward/outward rotation and abduction (valgus/varus). Experiments demonstrate that both motions take place during normal walking. However, there are difficulties with measuring these motions using normal marker-based

motion capture methods – the measuring error can exceed the range of the measured motion [Reinschmidt, 1997]. This high measuring error makes it useless to include these two DOFs in the model.

Synopsis

- All three rotational DOFs of the knee have their roles in walking. However, the axial and lateral motions have very small amplitudes and it is almost impossible to measure them using standard marker-based MC methods. Therefore it is not practical to include these two DOFs in a walking model that is based on MC data;
- The knee flexion cycle can be described as a pair of peaks or parabolas. The first (small) peak is not very distinctive; its shape varies significantly among different gaits. This peak occupies the first 35-40% of the cycle and has a maximum value of 20-30 degrees. The second peak has a very smooth and distinctive parabolic shape; the parabola reaches its maximum of 75 (+/- 20) degrees at midswing (70% of the cycle);
- Similarly to the hip flexion curve, the knee flexion curve has a point of inflexion (maximum of the velocity) at about the time of *toe-off*. Though these two points of inflexion do not necessarily coincide with each other (and with the time of *toe-off*) precisely they are usually very close to each other;
- The normal range of knee flexion in walking is about 75 degrees. However this value can vary significantly in both directions, reaching a value of 120 degrees or going down to 50 degrees. An insufficient amount of flexion may disrupt the normal walking pattern, resulting in so-called circumduction gait;
- The knee does not necessarily fully extend during walking. This fact may present some difficulties for an MC importer, which usually does not have any other references to a full extension position of the knee.

3.5.3 Ankle and Foot

Role of ankle joint in gait

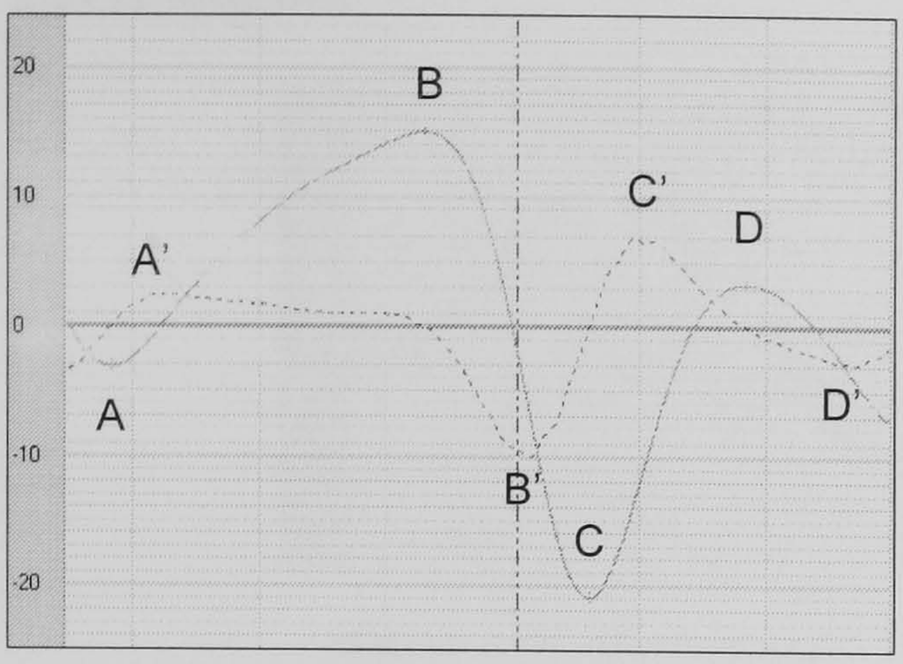


Figure 3.17: Ankle dorsiflexion curve: angle and velocity (dotted)

Stance phase: The motion of the ankle during the stance phase is usually described in terms of three *rockers* [Ayyappa, 1997]. The first rocker (or heel rocker) begins at *initial contact* (*heel strike* for the normal gait) and ends at *foot flat*. When the foot hits the ground, the external momentum (Figure 3.17) produced by gravity forces the foot to plantar-flex from its neutral position to about 15° of plantarflexion. The purpose of this phase is to decelerate the body's inertia at initial contact.

During the second rocker (ankle rocker) the tibia advances over the planted foot (Figure 3.18b) until its forward motion is stopped by contracting muscles at about 10° of dorsiflexion. At this moment the heel leaves the ground and the metatarsal heads become the pivots for subsequent motion. The forward progression of the tibia is accompanied by its internal rotation, which in its turn is transferred into supination of the foot.

The last rocker (Figure 3.18c) is initiated with *Heel Off* and finished when the foot pushes off to begin the swing phase. At the beginning of this rocker the ankle continues to dorsiflex until it reaches the maximum (15/20°) and then it steadily plantarflexes throughout the terminal stance and the initial swing phase. This phase is also the only period when the metatarsophalangeal (toe) joints are active: as the heel lifts up and the body weight is transmitted to the metatarsal head and the toe, the metatarsophalangeal joint dorsiflexes to maintain contact with the ground. The peak of the toe dorsiflexion occurs just before *Toe-Off,* after which the joints quickly plantarflex back to the neutral position.

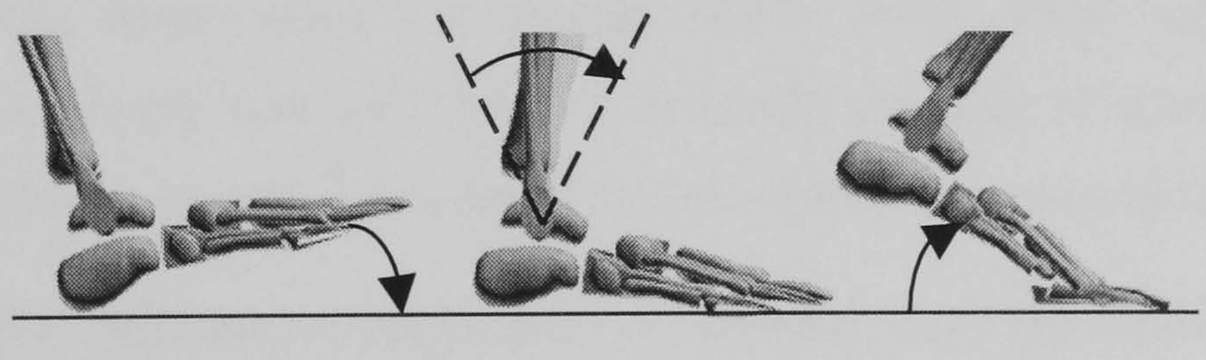


Figure 3.18: Three rockers of the foot: a) heel rocker b) ankle rocker c) forefoot rocker

Swing phase: During this phase, the ankle has two tasks: to help the knee with ground clearance and to prepare the foot for the initial contact. The first is achieved by returning the

foot from a plantarflexed position to a neutral position and keeping the foot in this position. Preparation for the initial contact consists of opposing the external plantarflexion moment so that the foot contacts the ground with the heel and drops to the ground not too quickly.

Curve analysis

Figure 3.19: Different patterns and ranges of dorsiflexion for two gaits

The ankle dorsiflexion curve has a more complex character and shows more variability than the hip and knee flexion curves. Figure 3.17 depicts the "standard" ankle curve on which two dorsiflexion and two plantarflexion peaks can be easily identified. However identification of the smaller peaks (A, D) becomes not so obvious (and sometimes impossible) if the gait deviates from the norm (Figure 3.19).

The larger peaks (B, C) usually can be found even in most unusual gaits and their automatic identification does not represent a problem: B corresponds to the maximum of dorsiflexion during a gait cycle and C corresponds to the next pronounced local minimum.

Figure 3.19 also illustrates the variability of the ankle's range of flexion. The normal range is between 20 degrees of plantarflexion and 15 degrees of dorsiflexion, though a large deviation (more than 100%) makes the "normal range" a very relative notion.

Effect of shoes on gait

Most biomechanical studies of gait are performed for natural "shoeless" walking. However, there is no doubt that shoe-wearing can considerably alter the gait pattern, particularly for the foot, and in extreme cases (when wearing high-heeled shoes, ankle-supporting boots, etc) it can result in a completely new gait. Since it is hardly possible to cover all the effects that different shoes produce on gait, we outline just a few basic features and their possible effects:

- Shoe flexibility: Stiffness of shoes can considerably limit the flexibility of the foot. To compensate for it. shoes are usually built with a slightly flexed sole. This affects the natural ankle movement: instead of going through three rocker phases the foot gradually rolls over the flexed sole and does not perform a proper push-off.

- Heels: In addition to disrupting the ankle-rocker mechanism, high heels make the gait less stable and therefore many aspects of the walking mechanism have to be modified to improve the stability.

Synopsis

- Among the three main joints of the foot complex (ankle, subtalar and metatarsal) the ankle joint has the most distinctive and important role in walking. The subtalar joint participates in walking too; however the measured pattern of its motion (inversion/eversion) is too vague, and most of the time the motion is passive. Nevertheless the motion in the subtalar joint should be included in the locomotion model since the ankle joint does not have enough DOFs for normal walking. The metatarsal (toe) joint is very important in bare-foot walking and less important in shoe walking.
- The ankle dorsiflexion/plantarflexion curve has a more complex and variable pattern than the hip and knee flexion curves. The curve that corresponds to normal gait is relatively smooth and can be clearly decomposed into four intervals where the curve is monotonic. However this behaviour is easily disturbed by small changes in walking pattern.
- In heel gait, the only period of the cycle when the metatarsal joint is active is in the last part of the stance phase, the *preswing* phase. And as the toe leaves the ground at the end of the stance phase, the metatarsal joint rapidly returns to the neutral position. In toe gait, however, this joint is active throughout the whole stance phase.
- The most prominent features of the dorsiflexion curve are the maximum and the minimum values of dorsiflexion in the cycle. The maximum value (20 degrees of dorsiflexion) is usually reached just after *heel-off* and the minimum value (15 degrees of plantarflexion) is reached during *initial swing*. Similarly to the hip and knee flexion curves, the point of inflexion of the dorsiflexion curve coincide with *toe-off*.
- In normal walking, the foot contacts the ground in three points (pivots): the heel, from *heelstrike* to *heel-off*; the metatarsal ball, from *flat foot* to almost the end of push-off; the toe, during the last part of *preswing*. In shoe walking, the positions of the pivot points and the duration of the contact are affected by the shape and flexibility of the sole.

3.5.4 Pelvis and Spine

As was said earlier, one of the most important locomotion-efficiency criteria is smoothness and small amplitude of the body's centre of gravity (COG) motion. Among the factors that control the character of this motion (gait determinants) are two motions of the pelvis: pelvis tilt and pelvis rotation.

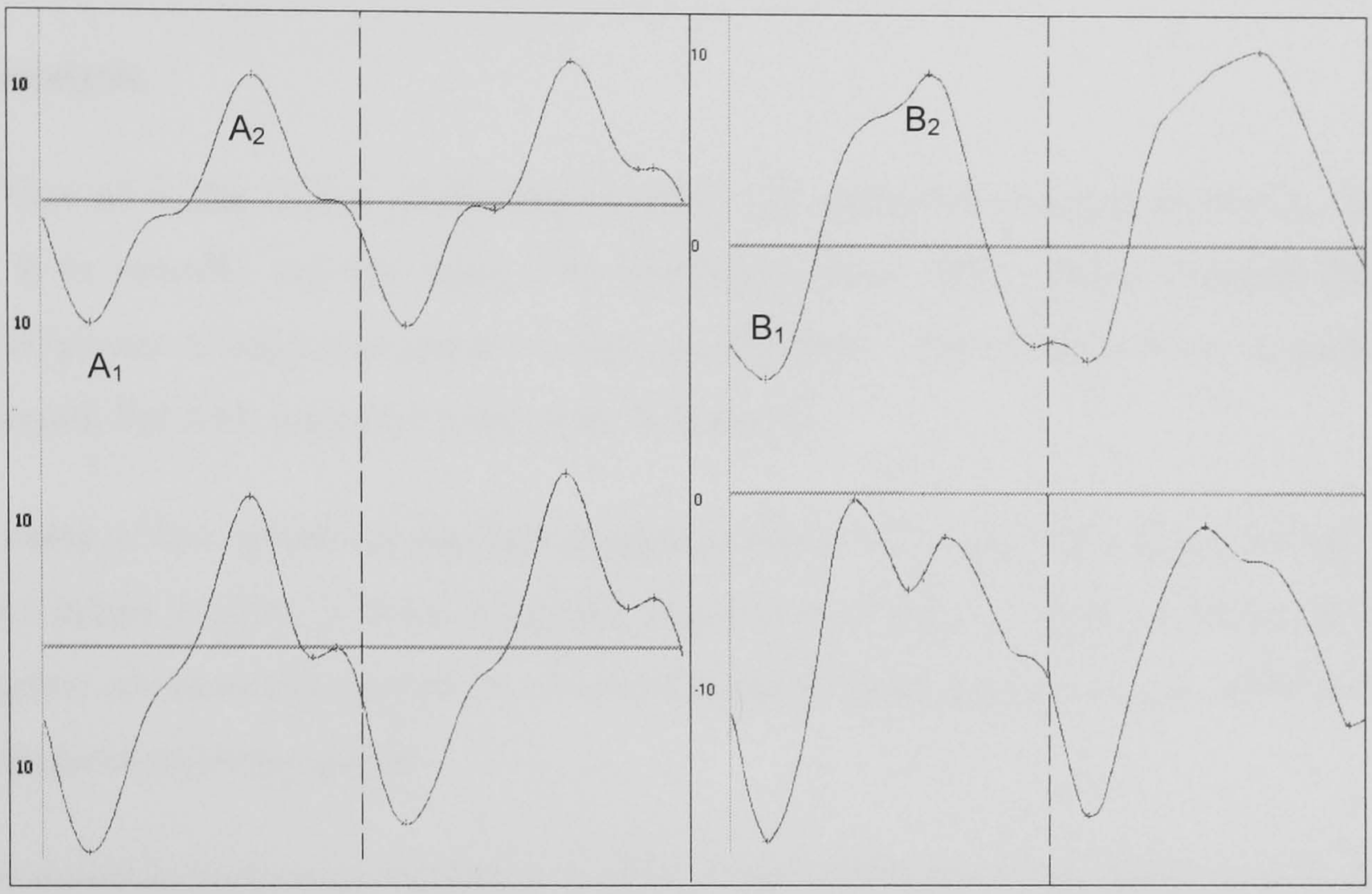


Figure 3.20: a) Lateral pelvis tilt and hip adduction curves; b) Pelvis rotation and hip rotation curves (vertical dashed line separates two strides)

Role of pelvis tilt

At about midstance, the hip of the stance leg reaches its highest point. Tilting the pelvis at this point (Figure 3.20a, point A) lowers the vertical position of the opposite hip and by doing so it also lowers the height of the COG, whose height depends on the average between hip's heights. Thus the amplitude of the vertical movement of COG is smaller than it would be if the movement of COG followed the movement of the hips.

Role of pelvis rotation

Pelvis rotation also helps to minimise the vertical excursion of the COG, but it does so by extending the stride length: it brings the hip forward when it flexes (B_1 on Figure 3.20b) and backward when it is extending (B_2). This means that less flexion and extension of the hip is needed and therefore the vertical movement of the COG is smaller.

As mentioned before, the movement of the pelvis is accompanied by the movements in the spine and hips. The motion of the spine replicates the motion of the pelvis very closely

compensating the tilting and rotating of the pelvis. The result of this compensation is that the trunk is always orientated in the direction of progression.

Analogously, the lateral and rotating movements in the hip compensate for the corresponding motions in the pelvis, allowing the extremities to move without deviations. The curves of these motions are also similar to those of the pelvis (Figure 3.17b), though the hip rotation curve is typically shifted in the direction of external rotation.

Curve analysis

On the face of it, the curves of the lateral pelvis tilt and pelvis rotation resemble sinusoidal curves. It is actually not the case. The lateral-tilt curve has a rather pointed shape and additional points of inflection about the time of *Midstance*. The rotation curve is more similar to a sinusoid, but with indistinct (and often forked) tips.

The extrema of the curves are reached at approximately the same time, about *toe-off*. At this point, the pelvis is tilted to about 8 degrees and rotated to about 5 degrees. Since the motions of the pelvis are normally symmetric, the total range of these pelvis motions will be about 16 and 10 degrees correspondingly.

The translational motion of the pelvis is even more useful for understanding and analysing walking than its rotational motions. The reason for this is that this motion is almost identical to the motion of the body's centre of mass, and so it can be equated with the motion of the figure as a whole. From the biomechanical point of view, this means that the motion can be used to evaluate the efficiency (or normality) of the locomotion, while, from the animator's point of view, this means that the motion is *very* important for the perception of the animation.

The translational motions of the pelvis in the horizontal and vertical planes have a strong sinusoidal character (Figure 3.21). The periods of these motions are equal to the gait cycle for the horizontal motion and half of the cycle for the vertical motion. In normal gait, the corresponding curves demonstrate great smoothness and do not have any local extrema apart from the maximum and minimum values, which are reached at about midstance (or double support period for the

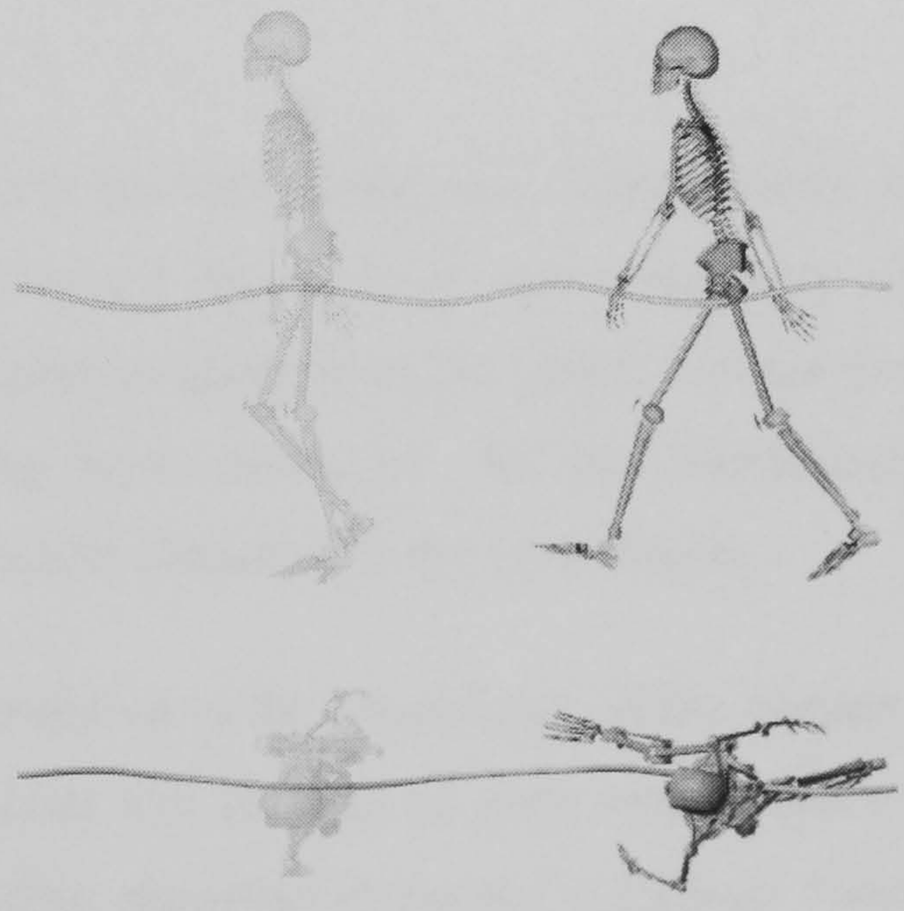


Figure 3.21: Vertical and horizontal movements of the pelvis

minimum of the vertical movement). In slow and pathological gait, the motions start showing irregularities, particularly the vertical motion, though the repeating pattern can be observed even in the most abnormal cases.

The position curve consists of two kinds of signals: signals representing small repeating deviations from the progression line and signals representing the progression of the body. Both types of signals have their uses in motion analysis but before either of them can be used they must be separated from each other.

Synopsis

- Two motions in the pelvis, lateral tilt and rotation, and the corresponding motions in the hip are a very important part of a walking mechanism. Unlike hip flexion and the motions of knee and the ankle, these are responsible not so much for the progression of the body as for improving the efficiency of the locomotion. The third motion of the pelvis, anterior-posterior tilting, does not have any particular role in walking.
- In normal gait, the motion curves of the pelvis have a relatively smooth pattern, which approximately resembles a sinusoidal curve. Both curves peak at the same time, at approximately the end of *double support*.
- To maintain the limb closer to the progression line, the hip counterbalances the lateral and axial movements of the pelvis. Therefore the curves of the hip and pelvis motions closely resemble each other, though there are some differences in the amplitudes and in details.
- The pelvis position curve is also important for the purposes of motion analysis. This curve has a distinct sinusoidal shape (in both the sagittal and transversal planes) and usually it is very smooth. The smoothness and the amplitude value of this curve can say a great deal about the efficiency and naturalness of the gait.

3.6 Summary

This first part of this chapter provided an insight into the biomechanics of human joints. It examined the major locomotion joints and demonstrated the diversity and complexity of motion in the real joints. Particular attention was paid to features of the joints that are not normally taken into account in models designed for figure animation. All this information formed the biomechanics foundation for the figure model discussed in the next chapter.

The role the major joints play in locomotion was analysed in the second part of the chapter. The aim of this analysis was to identify the character and features of joint motion and so create a background for the knowledge-based animation algorithm presented in Chapter 5 and Chapter 6.

4 Figure Model

The process of creating character animation consists of two parts: modelling the character and animating it. Following this division, the project was also carried out in two steps. The first step was to design and implement a figure model that would combine all the features needed for the procedural animation with the characteristics inherent to specialised biomechanical models. Then, on the second step, the figure model was used to incorporate biomechanical knowledge into the animation of human locomotion.

This chapter gives a detailed review of the figure model and explains some design solutions that were made during its development. The review starts with a description of the environment in which the model was implemented and used. This is followed by a description of the model architecture and an overview of its main elements. Some of these elements are closely examined in the following sections: *the link object* section discusses the structure of the figure model and its visualisation; *the joint controller* section concentrates on creating anatomically-accurate kinematic models for the joints of the figure; and *the animation model* section explains how the figure model interacts with procedural-animation algorithms.

4.1 Choosing the 3D Toolkit

Implementing a 3D-animation tool would be much more complicated if the developer himself had to tackle all the visualisation and interface issues. Fortunately, there are several graphics toolkits around that can make the development of 3D applications more efficient. While choosing such a toolkit for implementing the project, we considered the following alternatives:

- *OpenGL*: A low-level 3D graphics library. It supports rendering and texturing of a wide range of primitives and objects. Though this library allows the writing of highly efficient and portable graphics applications, it is not convenient to use and its functionalities are limited to low-level rendering;

- *Open Inventor*: This library provides a high-level object-oriented interface to OpenGL. It supports some high-level functionalities that are useful for figure animation. These include scene graphs, animation engines, etc;

- *3D Studio MAX API:* 3D Studio is a modern 3D modelling and animation package that provides a programming interface for extending its base capabilities. This API covers the functionalities of the above libraries and provides many other useful functions.

After comparing the pros and cons of these toolkits, the final choice was made in favour of 3D Studio MAX. The benefits of using this API outweighs its main drawback that the model can be used in 3D MAX only. These benefits include:

- The system takes upon itself most of visualisation and animation tasks allowing the developer to concentrate on modelling the motion;
- The modelling tools of 3D MAX can be used to create an environment for the animation. This and the possibility to skin the skeleton model using MAX plug-ins makes the evaluation of the motions more reliable;
- 3D MAX provides a state-of-the-art character-modelling tool, Character Studio. Comparing the animation produced by Character Studio and our system was extremely valuable for analysis and evaluation;
- The package has a large library of motion captured data. Considering practical difficulties with obtaining MC data, this library provided an invaluable resource for the project.

4.2 Overview of 3D Studio MAX

3D Studio Max is both a modelling and an animation system. It means that, in the same program, an animator creates 3D geometrical models, sets the scene lighting, puts materials to the objects and, finally, animates all this.

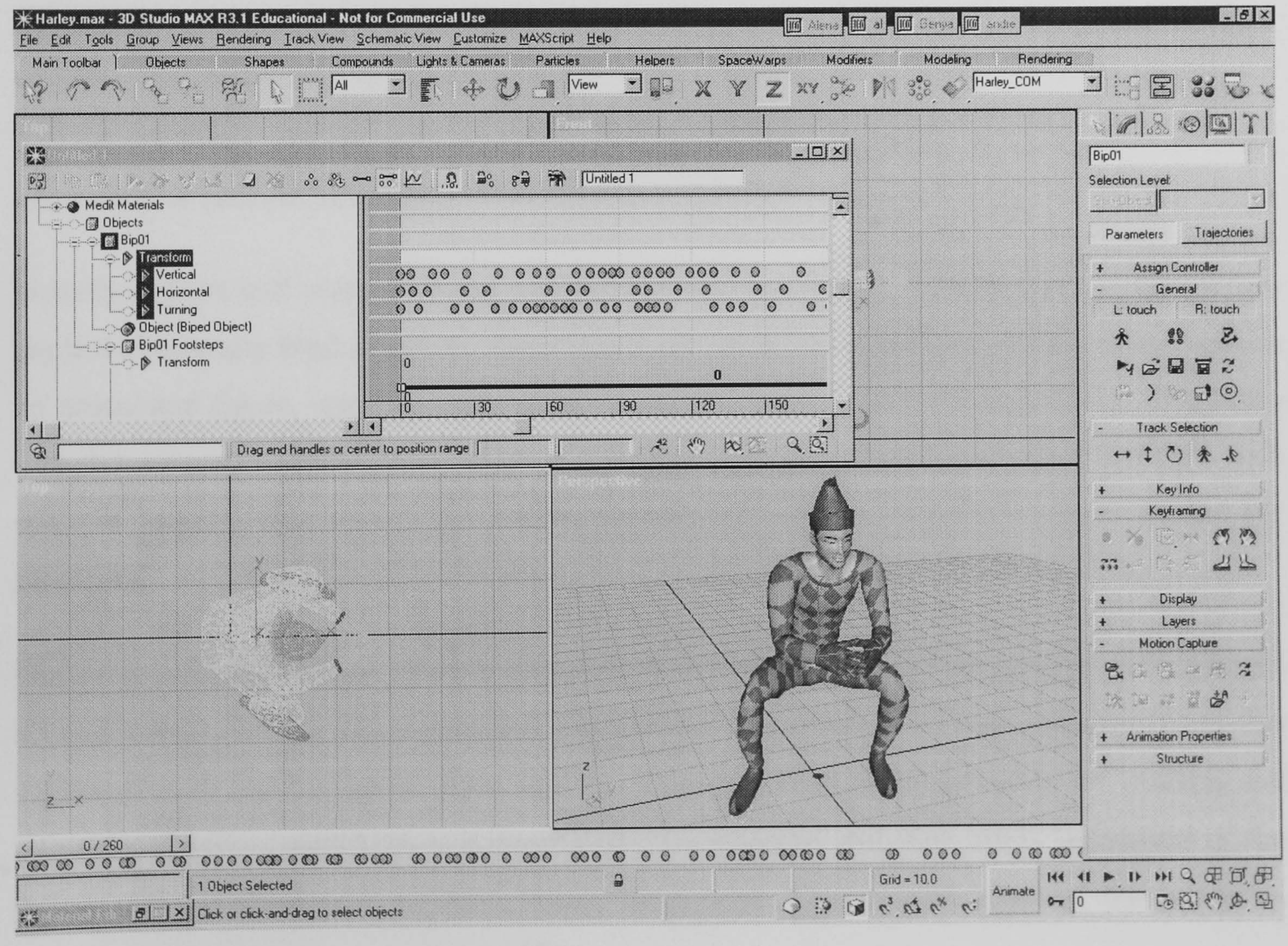


Figure 4.1: MAX User Interface

The program is based on object-oriented principles. Everything in MAX is an object: elements of a scene, sets of keyframes, user interface controls, etc. This approach provides a very flexible system in which new models and tools can be easily added or deleted from the environment, depending upon the animator's requirements.

The modelling part of MAX is based on procedural objects and modifiers. Procedural objects are mainly used to model objects of the real world. 3D Studio MAX provides many standard kinds of objects, for example: 3D geometric primitives, spline-based objects, meshes, Constructive Solid Geometry objects. Also, it provides means by which these objects can be modified in many ways (using modifiers) and by which compound objects like hierarchical skeletons can be created. Of course, as for any other kinds of MAX objects, new procedural objects can be programmed using the MAX SDK or the MaxScript language and added to the system.

Animation is supported in 3D Studio Max by means of controllers. Controllers are another type of MAX object and are used to control the change of the scene state over a time interval or as a result of some events. MAX provides different kinds of controllers: it can be a simple Float controller that interpolates a float value (for instance, rotation angle) using a set of user-defined key values, or it can be a procedure LookAt controller that ensures that a camera is always directed at a particular object. Moving objects are animated using transformation controllers. The standard transformation controllers are implemented using sub-controllers, where scalar float controllers are assigned to each degree of freedom (XYZ components of translation, Euler angles, etc), and the main controller combines the data from the sub-controllers to reconstruct a single transformation matrix.

Usually objects and controllers are independent of each other, so controllers can be used to control almost any kind of object. However, if one needs to control a complex object such as an articulated figure, standard multi-purpose controllers turn out to be inadequate. Animating complex systems requires close integration between the system and controllers. In MAX such systems (objects plus one or several specialised controllers) are called system objects (or plug-ins).

Our system for animating human locomotion is an example of a system plug-in. It consists of a combination of different objects and master-slave controllers. The main components of this system plug-in are the figure object and its controller. They are responsible for creating and editing a skeleton model (as a hierarchy of Link objects) and they allow animation of the figure using high-level controllers (track managers). The second layer of the model is

represented by Link objects (bones) and joint controllers, which replace the standard Euler (i.e. rotational) controllers to ensure that the motion is anatomically correct.

4.2.1 3D Studio MAX Software Development Kit

From the developer's point of view, 3D Studio MAX can be thought as an operational system for 3D graphics and animation. It consists of the kernel and a variety of plug-ins, some of which are native and some of which were added by third-party developers.

As mentioned before, 3D Studio MAX is based on the object-oriented paradigm. This is also true for its SDK. Its hierarchy of MAX classes reflects all kinds of MAX objects. When you are programming a new type of object, classes are inherited from the corresponding superclasses (Object for procedural objects, StdControl for controllers, Atmospheric for atmospheric effects, etc).

Interaction between the system and plug-ins is provided using the mechanism of virtual methods. For instance, when you implement a new type of procedural object you should provide appropriate implementation for some virtual methods inherited from superclasses of your class; thus you may write methods that create and initialise your object, provide its mesh representation, save and load the object, etc. The same mechanism of virtual methods is used to allow plug-ins to call MAX methods. When MAX calls a plug-in method, one of its parameters will be an interface object, which virtual methods are actually MAX system methods.

Plug-ins are added to the system independently. However, when one works in 3D Studio MAX one works with a unified system rather than with a set of isolated tools. This is because MAX objects can communicate with each other. This inter-object communication is handled using a mechanism of references. A reference is a system record about inter-dependencies between objects. The system uses this record to notify the owner of the reference about changes in the target of the reference. Consider an example in which a deformable skin is attached to a skeleton model. How would a modifier that deforms the skin mesh find out about changes of the skeleton posture? The answer is simple: the modifier holds a reference to the skeleton model, so it will be notified when the posture of the model has changed.

4.3 Architecture of the Model

The use of the 3D MAX SDK suggests a particular character for the architecture of the model. In particular, it implies the adoption of the object-oriented approach and of designing the system in terms of 3D MAX objects: plug-ins, objects, controllers, etc.

It was already noted that the figure model and the high-level figure-locomotion controllers are implemented as system plug-ins. This means that they consist of many objects and controllers, which are closely integrated with each other to allow modelling of complex motions. The diagram below gives an overview of the workflow between the main components of this system.

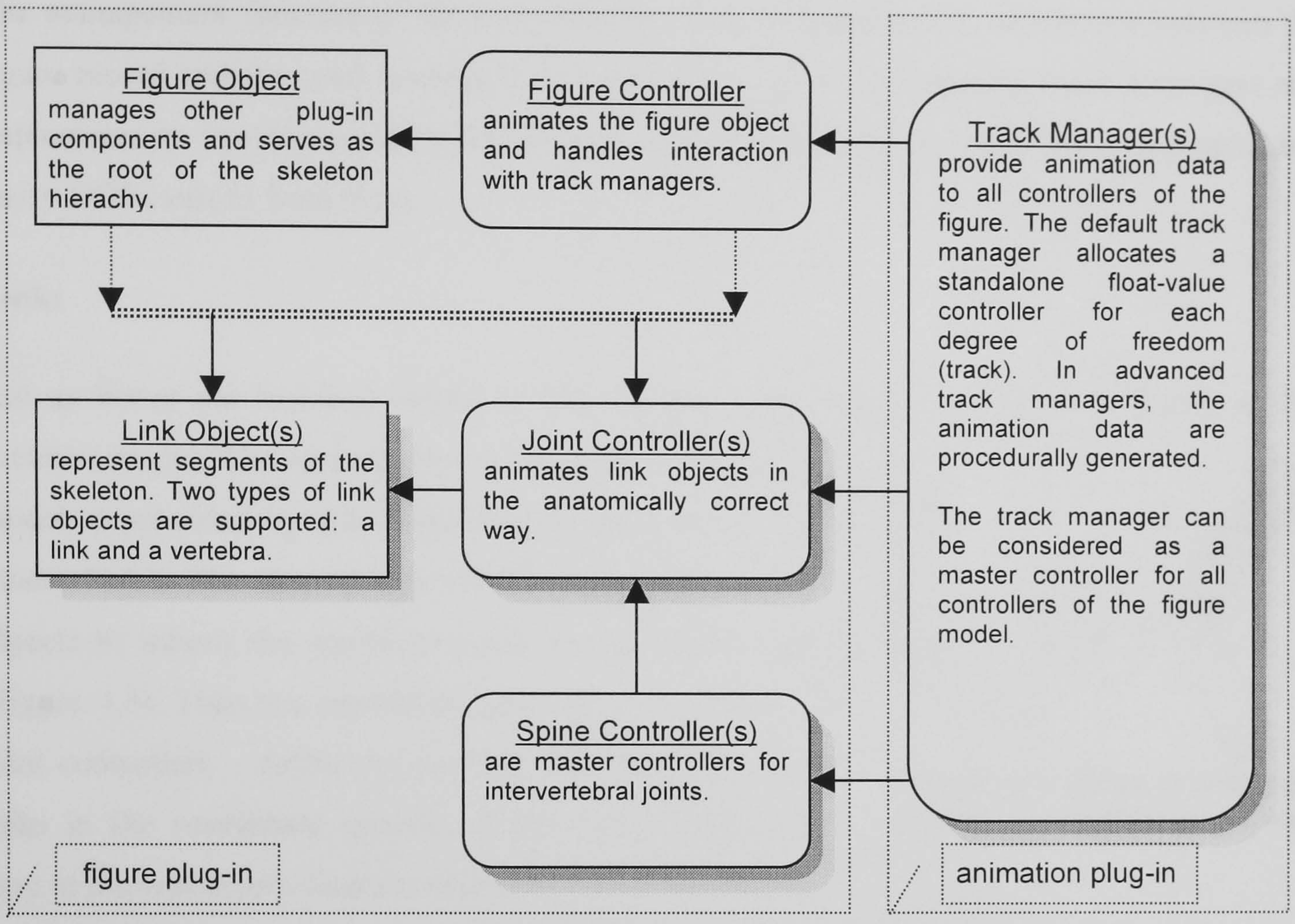


Figure 4.2: Animation workflow in the figure model

Figure

The primary function of the Figure object is to integrate all components of the model (links, joints, etc) into one system. The Figure object links all of these components to each other and handles their creation, initialisation and storing. It is also responsible for handling user interaction, so the model appears as a solid system rather then a collection of individual objects and controllers.

In addition to its management function, the Figure object serves as the root of the skeleton hierarchy. However, in contrast to other objects of the hierarchy, it is animated using a special Figure Controller not a Joint controller.

Figure controller

This controller is used to animate the Figure object, the root of the skeleton hierarchy. The motion of the root has six degrees of freedom describing progressive and rotatory motion along three orthographic axes.

The management function of the controller consists of handling the interaction between the figure model and the track managers. It maintains a list of the existing track managers and implements an interface used by the figure's controllers to identify active track managers and query DOFs values from them.

Links

Just as bones are building blocks of the skeleton, link objects are building blocks of the skeleton model. The connectivity of the links in the skeleton model is provided by the object-linking mechanism of 3D Studio MAX. The idea of object-linking is to allow children objects to inherit the transformations of the parent objects (Figure 4.3). Thus the controllers attached to the links – the joint controllers – define the position and orientation of the links in the coordinate systems of the parent links, rather than in the world coordinate system.

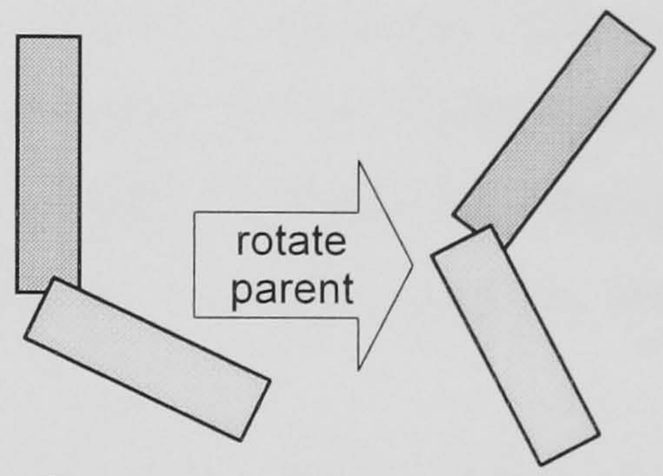


Figure 4.3: When rotating the parent the child rotates as well

As in other MAX objects, links must be able to render themselves. This is implemented by providing a polygonal mesh for each of the links. The meshes are read from a special file and scaled in accordance with the figure's height and proportions. Since the polygonal models are not hardcoded, but stored in a separate file, they can be easily modified or replaced. By default, a skeleton mesh is used to visualise the figure.

Joints

If link objects define the structure of the skeleton model, the joint controllers define how the model is animated. In 3D Studio MAX, linked hierarchies are normally animated by assigning an Euler rotation controller to each non-root object. In this model, the standard rotation controllers are replaced by the *joint controllers* in order to ensure that the motion is anatomically correct and to provide the controllers with specialised functionalities. Unlike the figure controller, the motion models for the joints are not predefined. So, by changing joint parameters (modifying the orientation of an axis, for instance) one can achieve a different behaviour of the model. Because the type of the joints and the number of degrees of freedom

are hardcoded in the program, it is not possible to change these from the user interface. However, such changes are practically possible and were occasionally performed for experimental purposes.

Spine controllers

The idea of spine controllers is to decrease the number of free DOFs in the figure by exploiting some dependencies in the intervertebral motions. The spine controllers transform the values of its DOFs into values for slave intervertebral joints. Since this transformation is based on biomechanics knowledge, the approach both ensures biomechanical correctness of the motion and compactness of the figure model.

Track managers

Track managers act as high-level master controllers whose purpose is to model complex figure motions. In contrast to the joint controllers, which are normally unaware of each other, a track manager controls all of the figure's DOFs, making it possible to model such highly integrated motions as walking. Track managers are implemented in separate plug-ins and dynamically assigned to animate the figure.

4.4 Link Object

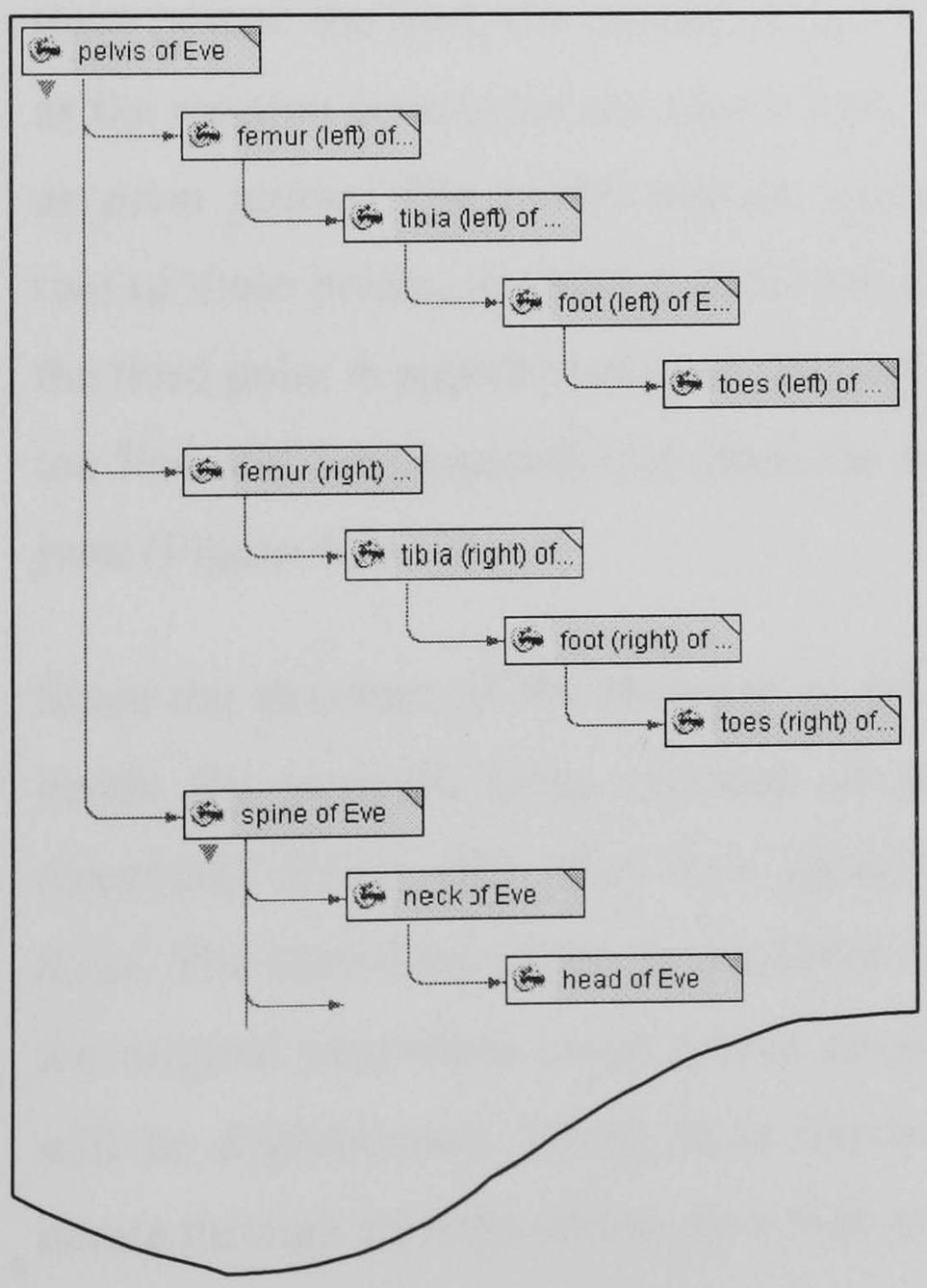


Figure 4.4: A part of the hierarchical figure model

A link object is the main structural element of the skeleton model. The model is constructed by connecting the link objects into a hierarchy and assigning a joint controller to each of the objects. So the links define the structure of the model, and the joints control its motion.

The skeleton hierarchy has a tree structure (Figure 4.4): each link has one parent, from which it inherits the transformation. The root of this hierarchy is the figure object; the leaves are the end-effectors (toes, hands and the head). In total, there are 21 normal links and 25 vertebrae.

Each link has its own coordinate system in which

positions and orientations of the children links (joints) are defined. For instance, the centre of knee rotation is defined in the local coordinate system of the femur link. If the link is scaled it automatically adjusts the positions of its children.

Visualisation of the link objects is implemented by providing each link with a polygonal mesh model. The models are loaded by a plug-in from a special file located in the configuration directory of 3D Studio MAX. The plug-in parses the geometry objects stored in the file (in ASE format) and associates them with the links in accordance with a simple naming convention (for example, an object with the name 'lfemur' or 'left femur' refers to the left femur link). This approach makes it easy to change the skeleton model if needed. By default, a skeleton-type model is used (Figure 4.5).

Figure 4.5: Default skeleton model

The link's display procedure provides a special interface that can be used to visualise track-manager data. For instance, the MC-based track manager, which performs automatic identification of foot constraints, highlights the constrained links.

An advanced animation algorithm needs more information about the skeleton model than just its structure and the offsets of its joints. In particular, it is important to know which parts of the foot contact the ground. In this model, it is assumed that the foot can contact the ground at three points: the heel, the metatarsal heads and the toe end. Since these points can be viewed as the rotation centres for the stance foot, they are referred to as *pivot points*. The model allows explicit specification of two of these points, the heel and the toe end. The position of the third point is approximated as the intersection of the heel-toe line and the perpendicular from the centre of metatarsal joint (Figure 4.6).

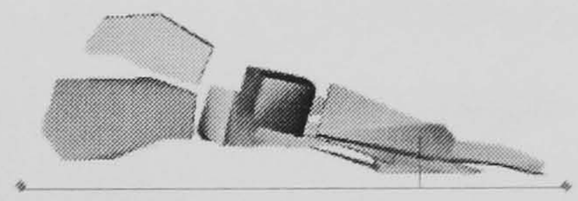


Figure 4.6: The three pivot points of the foot

Since the structure of the skeleton model is predefined, it is possible to reference the links inside the program using constant identifiers. There are thirteen base identifiers: *pelvis (rootLink), femur, tibia, foot, toes, spine, neck, head, mandible, clavicle, humerus, ulna* and *hand*. The identifiers of the symmetrical links (right femur, right tibia, etc) are derived from the original identifiers using special methods; for instance, the identifier of the right femur will be *Right(femur)*. Using these identifiers, it is straightforward to program cycles that iterate through all links and to store link-specific data in arrays.

4.5 Joint Controller

4.5.1 Choosing the Representation for Joint Motion

3D Studio MAX offers two approaches to representing rotations: using quaternions and using Euler angles. The quaternion-based representation implemented in Quaternion controllers boasts a natural and smooth interpolation between two rotations. On the other hand, these controllers do not allow the individual degrees of freedom in the quaternion to be intuitively displayed or adjusted. The representation used in Euler controllers exhibits the opposite properties: curves of individual rotation angles are descriptive and can be easily adjusted, whereas the interpolation of Euler angles is not as smooth as quaternion interpolation and may result in artefacts. Due to these restrictions, the choice of using a particular representation (controller) should be made in accordance with the requirements of the application.

The key requirement of the joint controllers is to provide a biomechanically accurate way of animating the skeleton model. However, neither of the standard controllers seems satisfactory for this purpose. Euler controllers represent motion as consequent rotations about the basis axes, which is not anatomically correct (Section 2.1.1). Using quaternion controllers is problematic too: quaternions do not provide any means to limit the range of motions to anatomically correct motions and do not allow manipulations involving individual DOFs. Therefore, specialised joint controllers, which extend the functionality of the standard controllers, should be created. The question is which of the two approaches to representing rotations should be extended: the quaternion approach, where the motion is represented by a single rotation about an arbitrary axis, or the fixed-axis approach, where the motion is represented by multiple rotations about predefined (static or moving) axes.

This table on the next page summarises the pros and cons of these two models in the context of joint motion modelling.

	Quaternion (helical axis) representation	Fixed-axis representation
support joint-motion constraints	-+ Although it is possible to incorporate motion constraints into quaternion-based model, it can significantly complicate the animation algorithms.	+ Most of the joint constraints can be expressed in terms of joint axes.
compatible with representations used in biomechanics	-+ The quaternions are widely used in biomechanics applications as an internal representation.	-+ The fixed-axis representation is the format used in the literature.
no loss of information / no ambiguities	+	- One and two-DOFs models cannot represent an arbitrary rotation. Though it is an advantage as far as the motion constraints are concerned, it may also turn out to be a drawback in some cases.
manipulation with individual DOFs	-+ A quaternion is defined by four values: three for the helical axis and one for the angle. Only one of these, the angle, can be considered individually.	+ Each DOF corresponds to the angle of rotation along one of the fixed axes. In most cases, the individual DOF provides sensible information and can be easily manipulated.
smooth analytic interpolation	+	-+ Individual axis curves can be successfully interpolated if the distance between control points is small. The problems start if the distance become large (keyframe animation) or the effect of gimbal lock occurs.

Table 4.1: Comparison of different representations of rotations.

Taking into account the importance of the first two properties, it was decided to adopt the fixed-axis representation for use in the joint controllers[1]. Thus, the concept of representing motion in joint controllers can be summarised in the following statements:

- A joint transformation is defined by one or several DOFs, which can be individually accessed or adjusted. The set of all the DOFs defines the pose of the figure completely;
- Each DOF represent an angle of rotation about a predefined axis. However, the application should not make an assumption that every DOF has a single axis associated with it: the actual parameters of the axis may depend on the DOF value (instant axis of rotation) or the axis can be abstract (for instance, axes of the spine controller);
- Joint motions are not limited to rotations only. Translation motions are also allowed.

4.5.2 Designing the Application Interface

There are a large variety of joints in the human body. It is practically impossible to create a single model to emulate all types of joints. Our approach is to allow different implementations of joint controllers, provided that they support a standard interface. In terms of object-oriented programming, it means that the parent class defines the base functionalities and allows the children classes to redefine some specific functionalities.

When designing such a class hierarchy, one has to compromise between generality and efficiency. The approach in which the interface is vague and most of the functionalities are implemented by the children classes does not put any limitations on the joint model. However, this approach is not efficient: not only should these functionalities be implemented for every class, but also the algorithms that use these classes become more complicated and implementation-dependent. A more efficient approach is to make some reasonable assumptions about the joint model that allow implementation-specific details to be hidden in the children classes. The interface presented is based on the assumption that the joint controllers represent the motion using a fixed-axis approach. The diagram 4.1 summarises the main methods of this interface:

---

[1] Evaluation of the joint controllers in the context of using MC-data demonstrated some drawbacks of this approach. The chapter 6 analyses these drawbacks and suggests some improvements to the model.

Class JointController
Forward-Kinematics methods (implemented in every subclass):
CreateMatrix(dof values) *DecomposeMatrix(input matrix, output dof values, error)* create/decompose a transformation matrix from dof values. These methods define the motion-model.
Controller-specific methods (implemented in the base class, can be redefined if needed):
GetValue(time) *SetValue(time, new value)* Inquiry/set the joint transformation at given time. These methods are used by 3D MAX to animate objects.
Inverse-kinematics methods (mainly implemented in the base class):
GetNumIkDOF() *ComputeDerivatives(dof, Jacobian, endEffector, currentTransformation)* *ApplyIncrement()* implement IK interface of the joint controllers. The default implementation is based on the assumption that the joint realises rotations about static axes, which have a single origin and whose orientations are specified in local coordinates. Any other kinds of joints (spine, for instance) should redefine these methods.
Parameter-manipulation methods (mainly implemented in the base class):
GetNumDOF() *DofByIndex()* *GetDOFRange()* Inquiry information about number of DOFs, their identifiers, etc. *GetPivot(dof), SetPivot(dof)* get/set the pivot point of the joint. If the joint has different pivots for different axes it has to define them relatively to the first pivot point. *GetAxis(dof), SetAxis(dof, axis)* *IsAxisConstant(dof), etc* get/set axis orientation for the given DOF. If the parameters of a controller cannot be inquired/set by these methods it should provide its own methods and an alternate user interface.
Other methods (mainly implemented in the base class):
User interface, MAX specific methods, etc

Diagram 4.1: JointController interface

As one can see, the core part of the interface, which defines the motion model, is very small. This makes it easy to add new joint controllers to the figure model. To do this, one just has to provide two methods: one that creates a transformation matrix from a series of DOFs, and one that decomposes such a matrix back into the DOFs values. The base class uses these methods to implement other functionalities of joint controllers. The programmer has to provide new implementations of these functionalities only if the standard implementations are not adequate.

Another important characteristic of the programming interface is that it allows other components of the model to deal with any joint controller in the same uniform way. This not only simplifies the implementation of these components, but it also makes them independent of the choice of the joint model and their implementation.

4.5.3 Types of Joint Controllers

This section describes five types of joint controllers that were designed to model joints in the figure model. These controllers include uniaxial, uniaxial with instant axis of rotation, biaxial, ball&socket and spine controller. Though this set of controllers may be inadequate to model all types of human joints accurately, it covers the joints that are most important for the modelling of locomotion, i.e. joints of the lower limb and the spine.

Uniaxial joint controller

This is the simplest joint controller in the model. It is used to represent hinge joints (the elbow) and joints that are approximated as hinge joints in the model (the toes). The uniaxial joint controller has only one degree of freedom: rotation about a fixed axis.

The transformation matrix *M* for this controller is created in three steps:

1. Apply a transformation T^{align} that aligns the joint axis with axis X
2. Perform a rotation about X.
3. Apply the inverse of T^{align}

$$M = \begin{pmatrix} x\cdot x\cdot(1-\cos\alpha)+\cos\alpha & x\cdot y\cdot(1-\cos\alpha)-z\cdot\sin\alpha & x\cdot z\cdot(1-\cos\alpha)+y\cdot\sin\alpha \\ y\cdot x\cdot(1-\cos\alpha)+z\cdot\sin\alpha & y\cdot y\cdot(1-\cos\alpha)+\cos\alpha & y\cdot z\cdot(1-\cos\alpha)-x\cdot\sin\alpha \\ z\cdot x\cdot(1-\cos\alpha)-y\cdot\sin\alpha & z\cdot y\cdot(1-\cos\alpha)+x\cdot\sin\alpha & z\cdot z\cdot(1-\cos\alpha)+\cos\alpha \end{pmatrix},$$

where α is the angle of rotation and (x,y,z) is the orientation vector of the axis.

The decomposition of the transformation matrix is performed in similar way:

1. Apply a transformation that aligns the joint axis with axis X
2. Perform Euler decomposition. The desired value is the first Euler angle (X).

Obviously, the decomposition is precise only if the joint axis and the helical axis of the original rotation are aligned. Otherwise, there will be a residual transformation $T^{residual}$ that should be applied to the joint transformation to get the original one ($T^{original} = T^{residual} \times T^{joint}$). This residual transformation is used to measure the accuracy of the decomposition: the transformation is represented as a rotation about a helical axis and the corresponding angle is taken as scalar measure of the decomposition error (Figure 4.7).

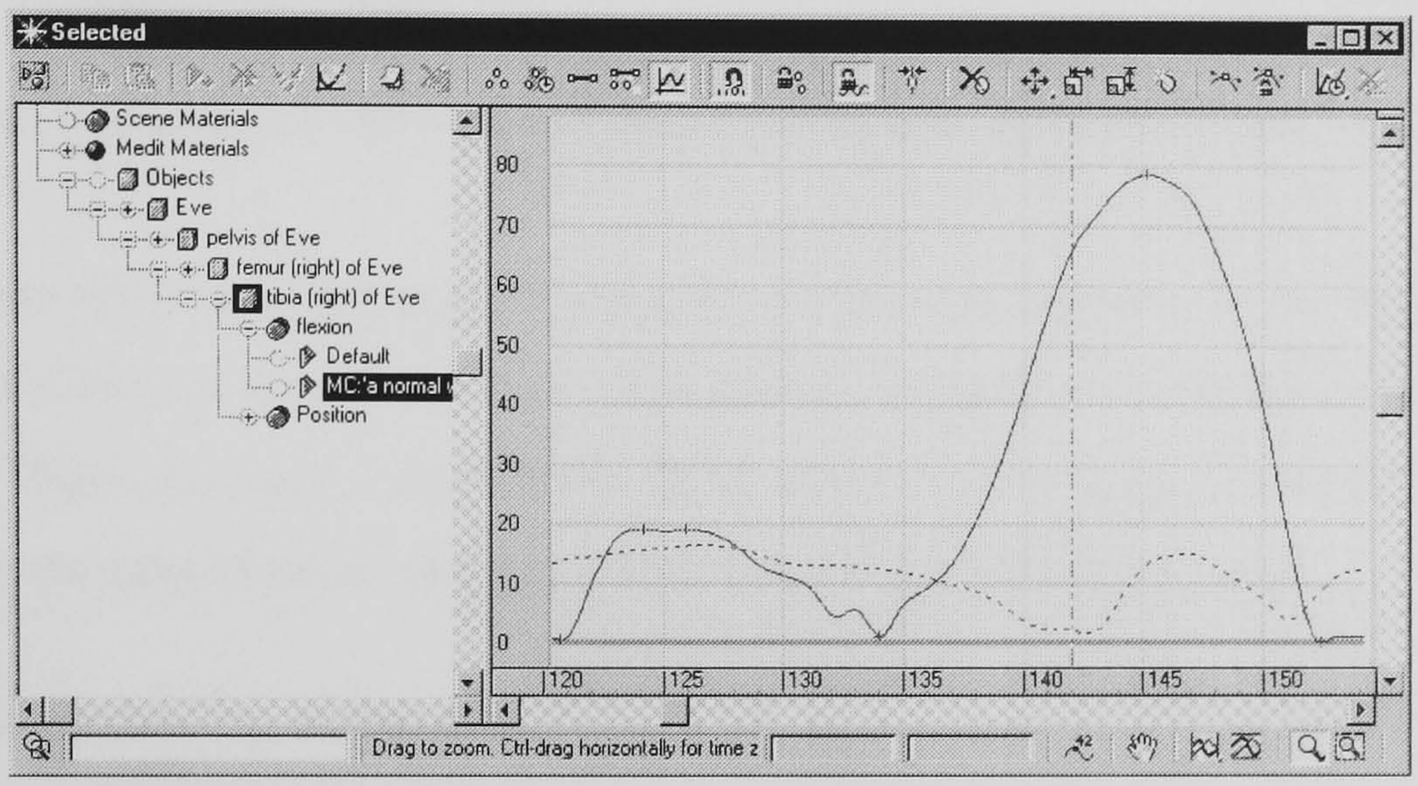


Figure 4.7: The dotted curve shows the error of representing the knee motion with a uniaxial rotation (the axis is not optimised).

Uniaxial joint with instant axis of rotation

This controller was specifically designed to model the knee joint. As stated in Chapter 3, the knee joint is not a simple hinge joint, and modelling the knee flexion as a rotation along a fixed axis is not anatomically accurate. In fact, knee flexion occurs as a combination of rolling, sliding and twisting motions of one bone on another. From the mathematical point of view this motion can be described as simultaneous rotation and translation along a moving axis.

The only way to model this motion is to use measured data. This data can be collected by a device called a goniometer (Section 2.2.2). This device samples the joint transformation while the motion is performed. After post-processing, the motion is represented as series of transformations (matrices), each corresponding to some particular period of the motion. These transformations can then be converted from the matrix form into a helical axis form, which finally gives us all the data needed to reconstruct the motion at any point. These include a set

of transformation matrices sampled at some points, rotation angles at these points (control points) and a set of axes that can be used to refine the motion between the control points.

The way the data are used to model the joint motion is demonstrated in Figure 4.8.

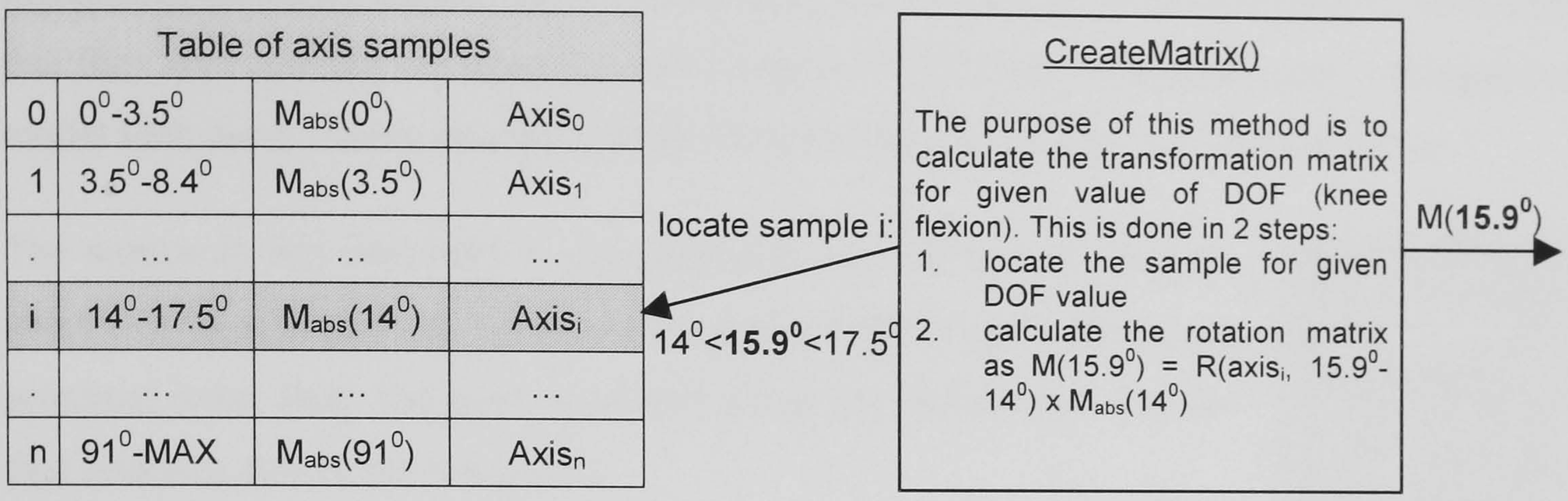


Figure 4.8: Motion model of the uniaxial joint with Instant Axis of Rotation

The inverse operation is implemented using an iterative procedure. First, the DOF value is estimated for the default axis. Then, this value is used to identify the measured transformation and the axis. These are used to refine the DOF value. If this value is not valid for the axis identified then the procedure is repeated once more for the new DOF value.

Because of the limitations of the goniometers, the measurements can be taken only when the limb is unloaded. Therefore, it is assumed that the joint motion is the same whether the limb is loaded or not. This assumption is however not exactly correct. The knee of a loaded limb does show a different motion pattern than a free-limb knee. This raises doubts about how accurate it would be to use these data in the situation where modelling of the stance limb presents the greatest interest.

Another problem with using the measured gonio data is how to transform the data from the reference system of the goniometer into the local reference system of the figure. Obviously, the approach in which the goniometer system is assumed to be aligned with the figure coordinate frame is not accurate. Thus it may reduce the positive effect of using highly accurate motion data.

Taking into account these facts, it was decided to model the knee joint using a fixed-axis uniaxial controller unless the analysis of the animation suggested that a more accurate approach would be beneficial. However, experiments have shown that highly accurate joint modelling is not crucial for an MC-based animation algorithm: the error contained in motion data is far greater than the potential error from using a simpler joint model.

Biaxial joint controllers

There are several human joints that allow motion in two planes around two axes. However, since these biaxial joints are not important for locomotion modelling, none of them was included in the figure model. On the other hand, there are joints located so close to each other that they are normally considered as one complex. The biaxial joint controller is designed to model such cases – more precisely, it models joint complexes with two uniaxial joints.

The motion in this controller is represented as two successive rotations about two fixed axes. First, the rotation is performed along the axis of the proximal joint; then, the joint is rotated along the transformed (by the first rotation) axis of the distal joint.

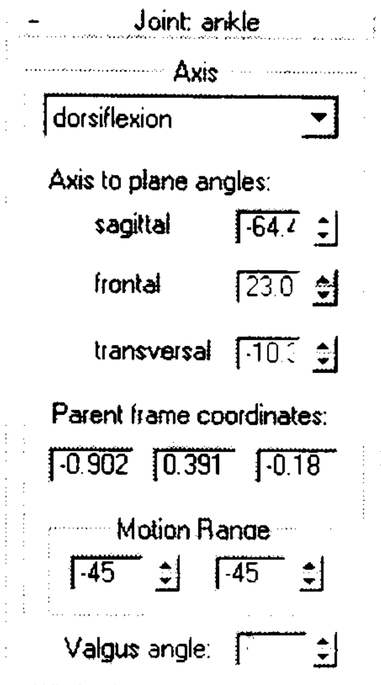


Figure 4.9: User interface for specifying joint parameters

There are no limitations on the orientations of the axes. The default orientations, which are taken from biomechanics literature (Appendix B: Joint Data), can be modified by the user through the joint-controller user interface (Figure 4.9).

The transformation matrix for this controller is created by multiplying matrices of two single rotations: $M = M^{distal} \times M^{proximal}$.

The inverse operation (matrix decomposition) is more difficult. An easy solution would be to consider the biaxial joint as two independent uniaxial joints. So, the input matrix would be first decomposed to extract the rotation about one axis, and the rotation for the other axis would be then extracted from the residual transformation. The approach, however, neither guarantees that the matrix creation and matrix decomposition will be mutually inverse operations, nor that the decomposition error will be minimal. The approach implemented in the biaxial controller is less straightforward: the required angles are calculated using the Jacobian Inverse technique (see Appendix A). Though this technique still cannot guarantee mutual invertibility in all cases, it guarantees that the decomposition error is minimal.

Ball & socket joint controller

This controller models motion in three-DOFs ball&socket joints, such as the hip or the shoulder. The real ball&socket joint permits any rotatory motion, and so does the controller.

The motion in the controller is represented using the classical Euler angles, which define rotations about the local coordinate axes. The order of rotations is predefined: first, rotation is done along X-axis (flexion), then along transformed Y-axis (abduction) and finally along

transformed Z-axis (rotation). Thus, the transformation matrix for the controller can be created by multiplying the matrices of the basis rotations: $M = M^{Z} \times M^{Y} \times M^{X}$:

$$M = \begin{pmatrix} \cos\beta \cdot \cos\gamma & \sin\alpha \cdot \sin\beta \cdot \cos\gamma + \cos\alpha \cdot \sin\gamma & -\cos\alpha \cdot \sin\beta \cdot \cos\gamma + \sin\alpha \cdot \sin\gamma \\ -\cos\beta \cdot \sin\gamma & \cos\alpha \cdot \cos\gamma - \sin\alpha \cdot \sin\beta \cdot \sin\gamma & \cos\alpha \cdot \sin\beta \cdot \sin\gamma + \sin\alpha \cdot \cos\gamma \\ \sin\beta & -\sin\alpha \cdot \cos\beta & \cos\alpha \cdot \cos\beta \end{pmatrix},$$

where α, β, γ are the rotations along X, Y', Z'' axes correspondingly

The decomposition of the matrix is realised by a standard matrix-to-Euler-angles decomposition procedure, which can be expressed (in simplified form) by the following expressions ($m_{i,j}$ are elements of matrix M):

$$\alpha = arctg\left(-\frac{m_{2,1}}{m_{2,2}}\right), \quad \beta = arctg\frac{m_{2,0}}{\sqrt{(m_{2,1})^2 + (m_{2,2})^2}}, \quad \gamma = arctg\left(-\frac{m_{1,0}}{m_{0,0}}\right)$$

Normally, Euler-angle decomposition is unique. The problems start when β, second rotation angle, approaches ±90°. In this case, referred to as *gimbal lock*, the X and Z axes become parallel. This effectively reduces the number of degrees of freedom and makes the decomposition ambiguous. As a result of this ambiguity, the Euler-curves become discontinuous and cannot therefore be properly interpolated. Since the effect of gimbal lock does not affect the joints of the lower body (due to natural restrictions on joint motion), it was not specially processed in the model. However, gimbal lock does affect the shoulder joint and it should be taken into account if accurate modelling of these joints is required.

Spine controller

Though this controller implements the same interface as the previous controllers, it is very different from them: the spine controller is a master controller that manages not only its own transformation but also the transformations of the slave sub-controllers (intervertebral joints).

Typically, the vertebral column is modelled with 3/4 links connected by joints with three degrees of freedom each. This provides adequate flexibility of the column without a significant increase in the number of DOFs, although this approach is not anatomically correct since it does take into account interdependencies between the intervertebral joints (Section 3.2.4).

By exploiting the intervertebral dependencies, the spine controller kills two birds with one stone: firstly, it asserts the anatomical correctness of the spine motion; secondly, it reduces the

number of degrees of freedom to the minimum. The following diagram illustrates the work of the spine controller:

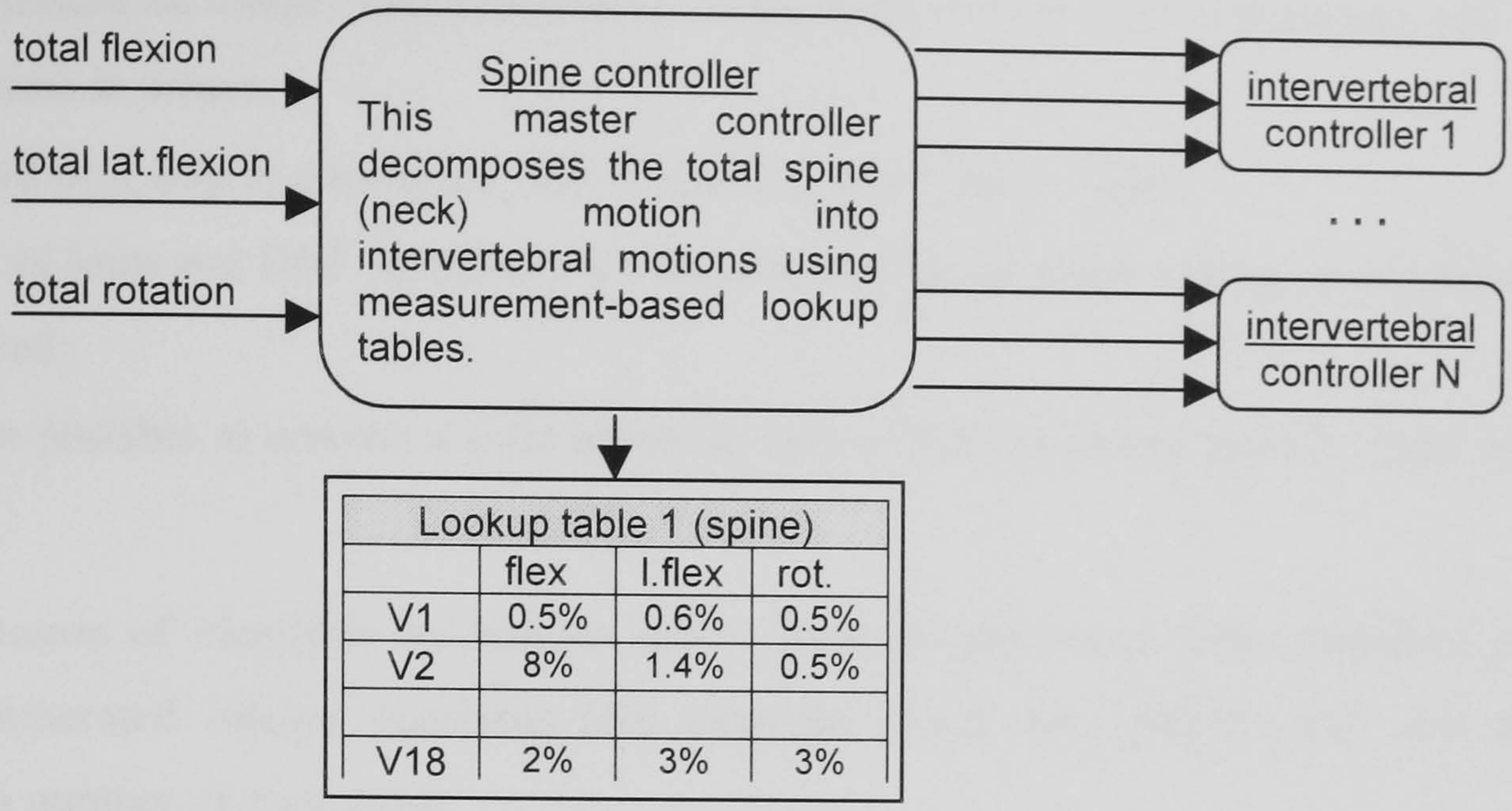


Figure 4.10: Spine controller

The lookup table (Appendix B) is the core of the spine controllers. It contains data of the relative flexibility of the intervertebral joints. The spine motion is distributed among the joints proportionally to their flexibility coefficients.

This principle of motion distribution works only if the spine (neck) motion is normal and physically unconstrained. For example, it would not work for a figure in a corset because the corset would affect the distribution of the flexibility. These restrictions, however, are applicable to specialised applications only (for instance, ergonomic design systems). For standard animation applications, the use of the spine controller produces natural and realistic results.

Accurate modelling of the spine has not only advantages but also has a few side effects. These effects are a result of complication of the model due to the high subdivision of the vertebral column (26 segments). Though this complication does not affect the kinematic parts of the figure model (thanks to the master-slave controller mechanism), it may affect some other part of character modelling such as the skinning algorithm. This algorithm requires the animator to specify which vertices are modified by the transformation of a joint. Therefore, the increase in the number of joints makes this procedure more time consuming.

4.5.4 Joint and DOF Notations

An important design issue for a figure model is how to implement a mechanism of addressing joints and their degrees of freedom. One solution is to associate each of them with a variable. This approach has some limitations though: one cannot use such variables to create iterations,

index data arrays, etc. Another solution is to assign a unique identifier to each joint and DOF. Such identifiers should meet several requirements:

- Identifiers should be integer and continuous, so that they can be used in iterations and to index elements in arrays;
- There should be a way to choose the side of a joint / DOF, left or right;
- The values of joint and DOF identifiers should not overlap, to allow misuse of identifiers to be detected;
- It should be possible to convert a joint identifier into a DOF identifier and the other way around.

Our implementation of identifier mechanism meets these requirements. The identifiers are defined as enumerated integer constants (for example, *knee, knee_flexion_dof*) and are provided with a number of operations:

Changing and inquiring the side of the joint/DOF:

```
Left(id), Right(id)
Left(id, boolean expression), Right(id, boolean expression)
IsLeft(id), IsRight(id)
```

Converting DOF identifiers into joint identifiers and vice versa:

```
JointId(dof), JointDOFIndex(dof), DOF(joint id, index)
```

Checking identifiers:

```
IsJointId(), IsDofId()
```

Accessing joints:

```
GetJoint(joint id), GetLink(joint id or link id)
```

Despite the simplicity of the described approach, it has proved to be very useful in the implementation of animation algorithms. In particular, the use of natural identifiers (such as *hip, knee, ankle_dorsiflexion_dof*) has resulted in the code being easily readable, which is extremely practical when one is implementing complex motion-analysis algorithms.

4.5.5 Analysis Tools

Since the figure model is intended as an experimental tool for modelling human motion, it is very important to provide a means for analysing various aspects of the figure motion. This is implemented by extending standard MAX tools for displaying data concerning the controllers. The joint controllers are provided with methods for plotting joint motion curves, visualising trajectories of joint centres and end-effectors, etc. They are also provided with a special interface that can be used to visualise information of the high-level animation controllers (track managers). For example, the MC controller displays results of motion analysis (Figure 4.11), approximation errors of one and two-DOFs joints, the amount of correction introduced by a retargetting algorithm, etc.

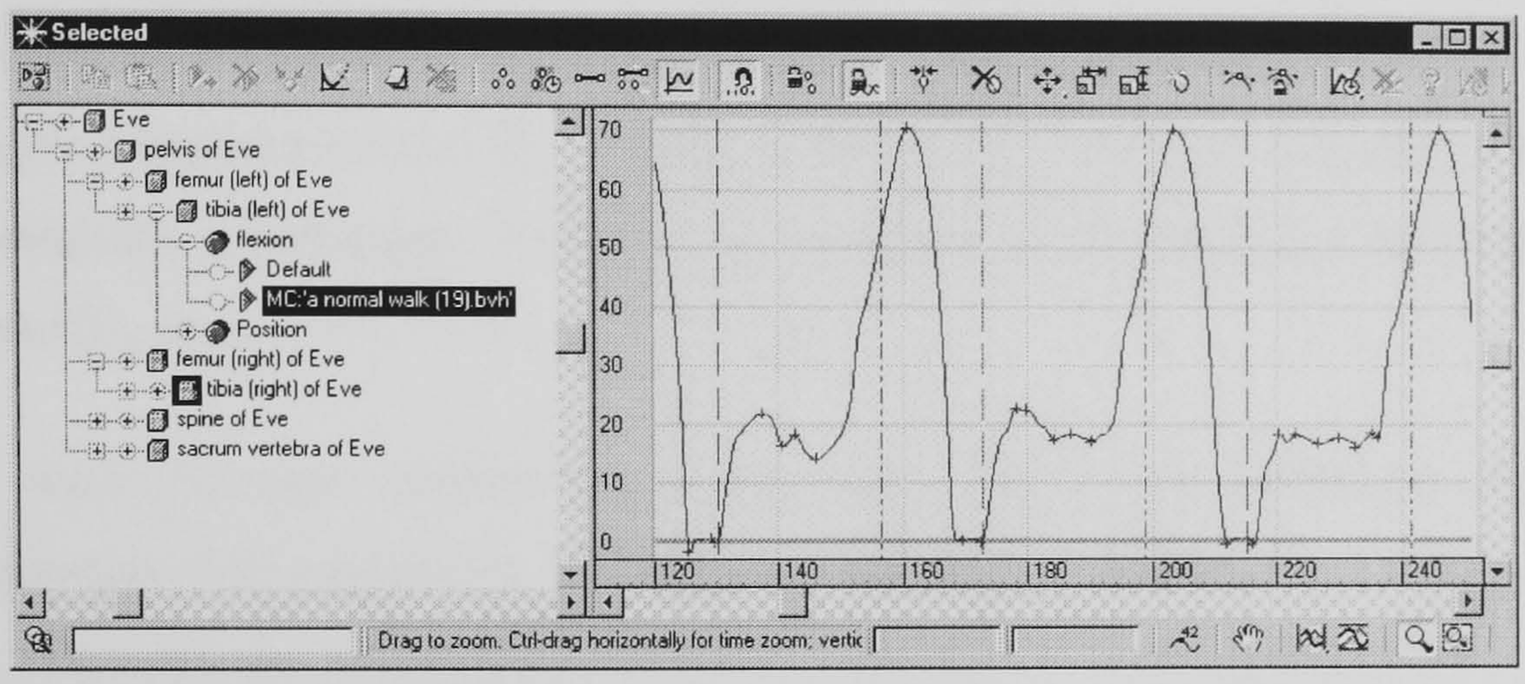


Figure 4.11: Visualising joint motion in the track viewer. The dotted vertical lines mark the borders of the stance and swing phases.

4.6 Animation Model

One of the design requirements of the model was to separate the structural and animation parts of the model. In this way, the same figure model can be re-used in other figure animation projects. To make this separation possible, the figure plug-in provides a special "animation" interface. The high-level animation controllers, *track managers*, use this interface to animate the model (Figure 4.12).

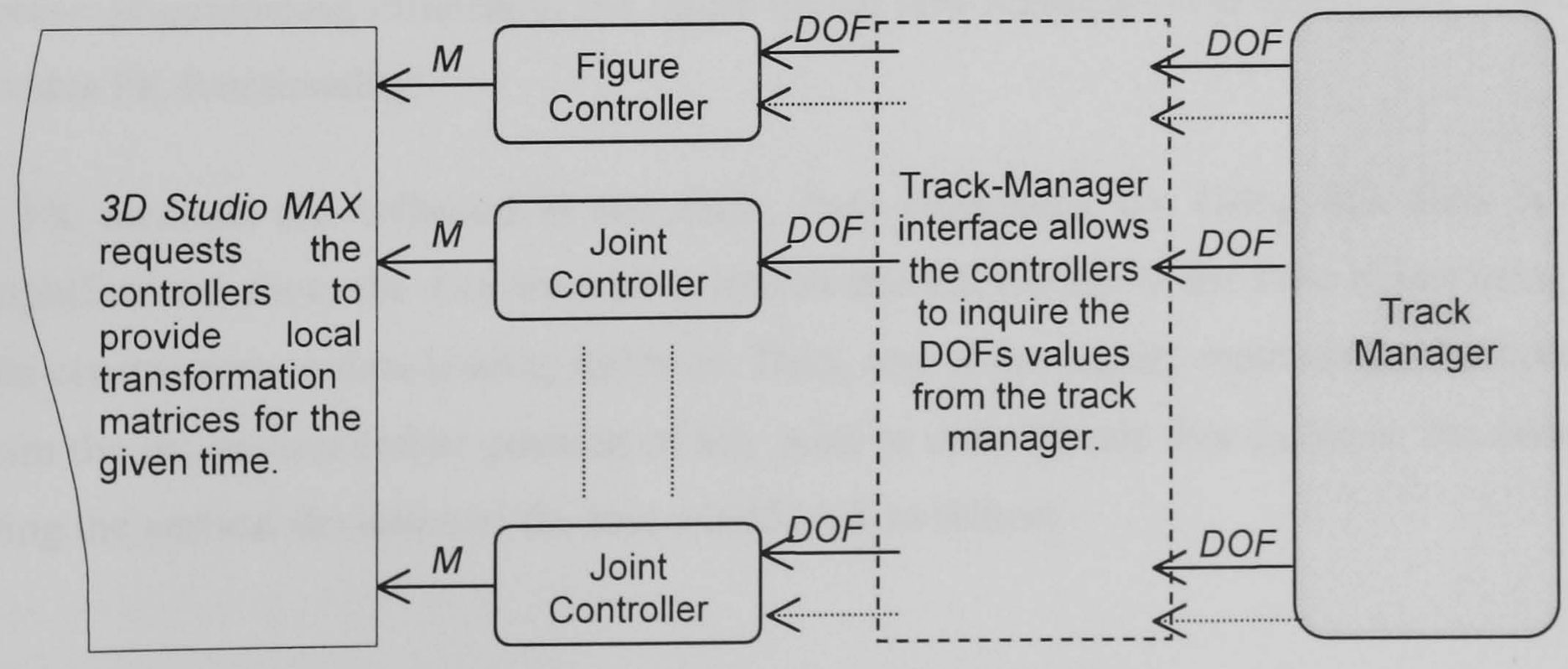


Figure 4.12: Animation data flow

The track manager is a master controller that provides the joint and figure controllers with the animation data. In the simplest case, the track manager just maintains a list of sub-controllers (tracks), one for each DOF. An advanced procedural track manager is responsible not only for storing the animation data but also for creating of the animation.

The assignments of track managers to the figure as well as other management methods are handled by the main figure controller. This controller also provides its part of the animation interface: it checks which track managers are active at any given time, inquires their values and combines them before returning to the controllers.

In scope of this project, two track managers have been implemented. The first one serves as an interface between the figure model and any standard MAX controller. By default, this track manager assigns a keyframe controller to each of the DOFs. This track manager is particularly useful for assigning a polygonal "skin" to the skeleton or for testing a particular figure's functionalities (IK, etc).

The second track manager implements a procedural MC-based animation algorithm. A detailed description of this algorithm will be given in the next chapter.

4.7 Simplifying Forward Kinematics

Reconstructing the world coordinates of links of a figure is one of the most common operations conducted with the figure model. The reconstruction is performed by traversing the tree of the joints and accumulating the local transformations. This algorithm (or Forward Kinematics algorithm) is used in almost every part of the animation workflow. 3D MAX takes care of Forward Kinematics (FK) for most of the standard tasks (modelling, visualisation, etc). As for the more advanced applications such as procedural animation algorithms, the developer has to implement the algorithm each time FK is needed. In order to improve programming efficiency, the figure model API (application programming interface) provides FK functionality.

All FK methods are collected in one class, *Pose* (Appendix C). Using this class is very straightforward. First, the data are read to into an internal storage of the Pose object using one of its constructors or data-loading methods. Then, one of the inquiry methods can be called to obtain the orientation and/or position of any joint or end-effector. For instance, the code for testing the vertical deviation of the foot would look as follow:

```
// load the current DOF data
pose.Load(GetCurrentTime());
// choose the lowest of two end-points, heel and toes
pos = (pose.GetHeelPosition(side).z < pose.GetToesPosition(side).z) ?
       pose.GetHeelPosition(side) : pose.GetToesPosition(side);
// calculate the deviation from the ground level
deviation = pos.z - GroundLevel(pos.x, pos.y);
```

Using the Pose class, the programs become much simpler. However, it may seem that the code also becomes less efficient. In reality, if one needs to know transformations of two or more joints in the same kinematic chain (as in the given example) one has to call FK methods several times. As a result, the same chain will be traversed several times and unnecessary calculations will be performed. This is not the case with the Pose class. The class supports a lazy caching scheme, which means that relative and absolute transformations for all the joints are remembered for further use. If some of the DOFs have changed, the affected joints will be marked and recalculated upon the next inquiry. This optimisation allows the programmer to concentrate on the algorithmic issues without worrying about efficiency of the produced code.

Besides the standard FK functionalities, the class provides some other methods that are particularly useful for motion analysis and correction: balance analysis, identification of foot constraints, correction of foot orientation, etc.

4.8 Summary

This chapter introduced the foundation of the practical part of the project, the figure model. The first two sections of the chapter explained why 3D Studio MAX was chosen as the base platform for the implementation and gave a short summary of this animation system. The actual description of the figure model started with an overview of its architecture. This was followed by description of all components of the model, with particular attention paid to the joint models. It was demonstrated how biomechanical effects discussed in Chapter 3 were implemented and how the biomechanics-based models developed were integrated into the main figure model.

The figure model introduced was designed to be universal, to enable it to be animated using different animation techniques. The following chapters, Chapter 5 and Chapter 6, demonstrate how this model was animated using a particular MC-based algorithm and evaluate the model in the context of this animation method.

5 Animation of Human Locomotion Using MC Data

As shown in Chapter 2, the only techniques that allow the production of a wide range of realistic animations with minimum user intervention are those based on using captured data. However, it was also demonstrated that simple playback of MC data is not a satisfactory method for most of the applications. This is due to the fact that the animations produced by such methods look realistic only if the proportions (not to mention structure) of the model are close to the ones of the performer; and this is not the case in our project. Therefore, to make MC data really usable, it is necessary to combine MC-based techniques with methods that can modify the motion in accordance with the animator's needs.

Since the major concern of the proposed biomechanical model is to improve accuracy and realism of animations, using MC data for animating the model comes as a natural choice. This chapter describes how MC was used to animate the model, i.e. how MC data can be converted into a form understood by the model and how problems that arise when uncorrected motion data are applied to an accurate figure model are corrected. Also, some typical problems of the MC approach (retargetting, motion blending) are analysed here in the context of using an anatomically accurate figure model.

5.1 Importing MC Data

This is the first, and the most straightforward, phase of a MC-based animation algorithm. It has however some critical points that should be taken into account to guarantee that the data are accurate.

5.1.1 Control and Data Flow

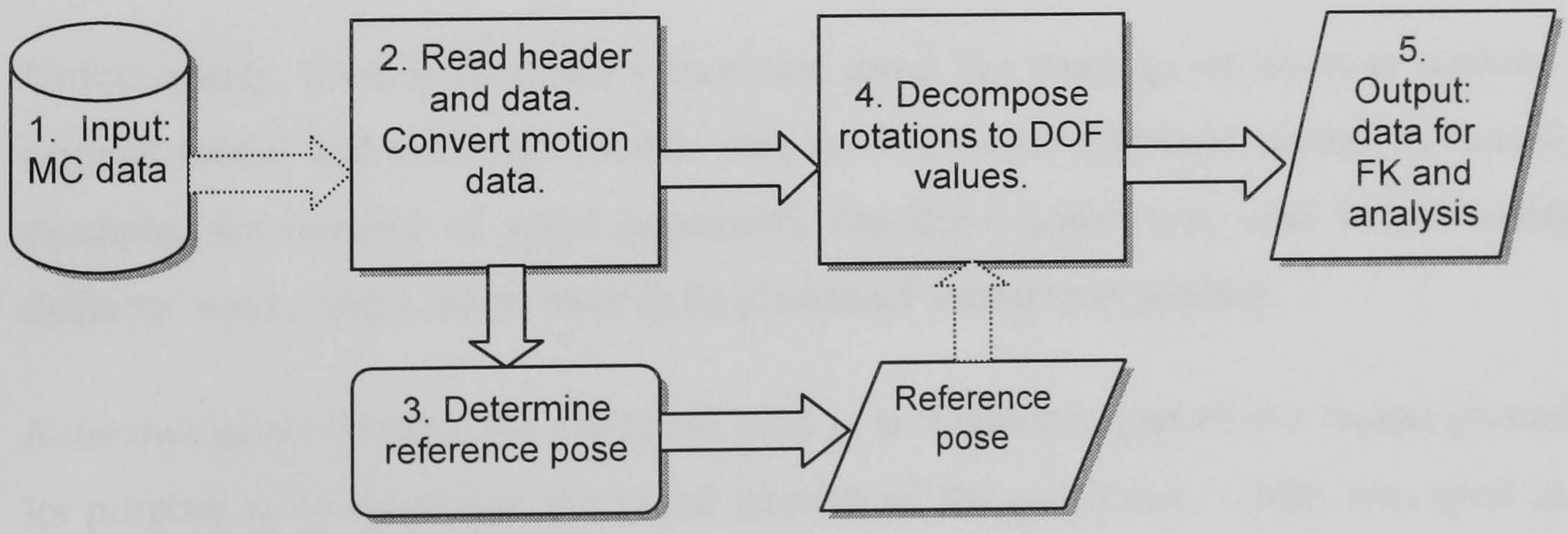


Figure 5.1: Control and data flow of MC import

The previous flow-chart represents the control flow of an MC import procedure. Here are the main stages of this procedure:

1. *Input data:* Though MC data come in various formats all of them can be divided into two groups: translational and rotational data. Translational data consist of series of 3D positions of markers. This kind of data needs to be pre-processed before the data can be used for animation, i.e. joint positions should be determined and joint rotations should be calculated. The result of this pre-processing is the data in the rotational format. Though the rotational format is obviously more convenient for animation purposes, it is less accurate[1] (since averaging process has been used) and thus translational data may be more preferable for some advanced applications.

   For the purposes of this project, an importer for BVH (Biovision Hierarchical File Format) files has been implemented. BVH is a popular ASCII file format supported by many applications. And, what is very useful, 3D Studio Max, the project's target platform, is supplied with a library of MC data files in this format (including more than 100 walking and running motions). Since this is a rotational file format, the original translational data, which are needed for motion analysis purposes, are reconstructed using Forward Kinematics;

2. *Loading:* The process of loading a BVH file consists of parsing the file header (Figure 5.2); reading the joint-motion data (usually these are given as series of Euler angles); converting this data to the internal format.

```
HIERARCHY
ROOT Hips
{
  OFFSET        0.00    0.00    0.00
  CHANNELS 6 Xposition Yposition Zposition
  JOINT Chest
    {
    OFFSET      0.00    6.27    0.00
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT Neck
     {.....................................................
```

Figure 5.2: Header of Biovision MC file. It defines model hierarchy, data channels (DOFs) and initial joint offsets.

   Unfortunately, there is no single convention about the topology of skeleton models. The original model and the target models may have not only different number of bones (for example, the number of spine segments), but these bones may also be connected in different ways. These issues have to be processed during data loading;

3. *Reference pose:* Finding the reference pose is an important part of the import procedure. Its purpose is to determine the initial posture of the performer, which was used as the

---

[1] Unless the measuring device captures rotational data directly i.e. using a goniometer.

origin for the transformations. If the posture is calculated (or defined) incorrectly, the accuracy of the data will be affected and as the resulting motion might not resemble the original;

4. *Decomposing rotations:* Since the input data for the model are values of degrees of freedom (i.e. pelvis tilt, hip flexion, etc.), rotational data (matrices or quaternions) should be converted to DOF format. Before converting quaternions to DOF values, they are corrected (i.e. multiplied by the correcting quaternion) to take into account the difference between the base (or zero) pose of the target model and the reference pose of the original model.

   The translational part of the data is simply scaled using a coefficient that represents the ratio between the dimensions of the performer and the model. This is not an accurate method and it allows propagation of errors. A more accurate method is used to correct the translational data during the correction phase;

5. *Output data:* The output of the MC data import are DOF values and reconstructed 3D positions of joint centres and end-effectors (heel and toe). These data are stored as splines to allow interpolation and data smoothing.

5.1.2 Identifying the Reference Pose

Any motion capture session always begins with setting up the performer in some reference position. This position is used to specify the character's initial set of coordinate systems, in which the joint transformations are defined[1]. Thus, any motion-capture data represent a set of **relative** rather then absolute values and the reference pose must be known in order to reconstruct the motion.

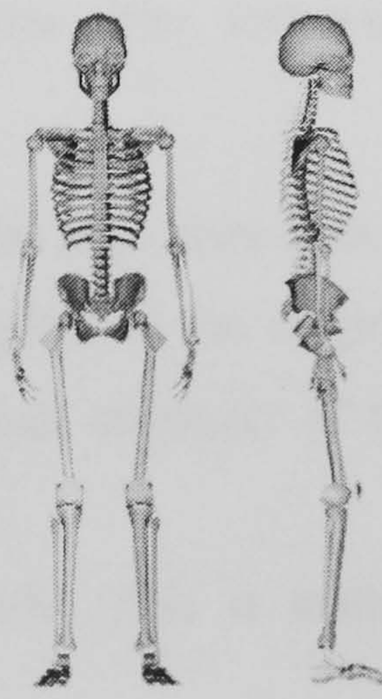


Figure 5.3: Rest pose

Normally, providers of the motion data indicate which reference pose they were using, so there is no need to identify it when using the data. The problem, however, still exists. Reference poses are not defined strictly: they can be described, but nobody can specify the exact joint

---

[1] It is assumed that the data are presented in the rotational joint-based format. If the original data are in the translational form (positions of markers) they should be converted into the rotational form.

angles corresponding to a particular pose. For example, a typical reference pose, *Rest pose* (Figure 5.3), is simply described as "a pose in which the actor stands at rest". Or, there is the *DaVinci* pose, which is defined as a posture with the arms horizontal and the legs spread apart. These vague approaches result in inaccuracies, and the animator then has to go through the laborious process of manual adjusting the pose.

An alternative approach is to calculate the reference pose by matching the zero poses of the reference and the target model. More specifically, positions of the joint centres of the reference model are matched with the ones of the target model. For instance, by matching the positions of the hip and the knees one can calculate the mutual orientations of thighs and estimate the hip joint transformation from one pose to the other. This method has some drawbacks, though:

- there are not enough data to calculate the initial transformation precisely; for instance, it is not possible to determine foot rotation from the positions of the limb's joints;
- there is no information in data files to set up initial transformation for the pelvis;
- this approach does not take into account constraints to the position of the figure, such as ground contact, and the joint limits.

Fortunately, there is no need to know the reference pose exactly. Human perception is tolerant of small changes in joint angles but, on the other hand, it is sensitive to the smallest artefacts in the motion. Therefore, instead of trying to achieve absolute accuracy, one can try to find a reasonable approximation to the pose and ensure that it will not produce any artefacts. Experiments with walking motions allowed some requirements of the reference position to be identified and suggested some modifications to the original algorithm. The following requirements have the most serious effects on the resulting motion:

- Feet should be in contact with the ground in the reference pose. If the algorithm fails to meet this requirement, the number of corrections that should be applied to the original motion may increase, making it more difficult to preserve the original character of the motion;
- The base position should not allow serious violation of joint limits. This is mainly applicable to the knee joint;
- The base position should be balanced. This affects the visual impression from the animation very noticeably.

To accommodate these requirements, the following algorithm was developed to find the reference position:

1. Approximate the initial position by matching the joint positions of the original and target models i.e. the "reference" rotation of a joint is the rotation that superposes positions of its child (distal) joint. For example, equations for the hip transformation are as follows: $R_{hip} = Rotation\ of\ asin(||Axis||)\ about\ Axis$, where $Axis = V_{femur}^{original} \times V_{femur}^{target}$ and V_{femur} is a vector with the start at the hip joint and the end at the knee joint;
2. Scan the motion file and check if the knee extension limit is violated. If the motion includes frames where the left/right knee is extended then the corresponding knee transformations can be used as the base ones. Since full knee extension is very typical for normal gait, this step allows to avoid clipping the motion.

   A simple heuristic can be used to understand when the knee reaches full extension: if the knee is in full extension it is locked and, therefore, it is more likely to stay in the same position for some period of time. The algorithm can check how long the knee retains its minimum position, and if this interval is above some threshold then it can be assumed that this minimum corresponds to full extension;
3. Calculate a pelvis transformation that puts the hip and ankle joints in the frontal plane. The spine rotation should mirror that of the pelvis. These transformations make a reasonable approximation to a balanced figure;
4. Adjust the ankle transformation to maintain proper contact between the feet and the ground[1]. This mainly has an effect on the amount of hip abduction/adduction, ankle plantarflexion/dorsiflexion and foot inversion;
5. Put the arms in one of the standard reference positions (horizontal, DaVinci, pose or vertical, Rest pose). The positions of the joints that do not participate in the motion correction process can be easily corrected after the motion import.

5.1.3 Decomposing Rotations

At this stage, the joint motion data are presented in matrix form. Since the skeleton model uses scalar DOF values as an internal motion-data format, all 3D rotations have to be converted into this format. The process of converting matrices (quaternions) into DOFs is straightforward: as shown in the previous chapter joint objects have a special interface

---

[1] The implementation assumes that the foot is aligned with the ground when the local Z-axis of the foot is orthogonal to the ground. In some cases this assumption can be wrong (for instance, if a character has high-heeled shoes). To detect such situations, one needs to analyse positions of the feet during motion.

(*DecomposeMatrix()*) for performing this task. However there are some issues specific for the biomechanics-based model:

- In general, the process of decomposing 3D rotations into DOF values is not invertible. The matrix calculated from the decomposed DOFs may be different from the original matrix. The difference between the original and the reconstructed matrices can be used to measure how accurate is the representation of a particular joint (number of axes, their inclination) for the given data set, which is useful for analysis purposes (Figure 5.4). There are several ways to calculate this difference in scalar form. We have implemented the following method: the difference matrix is considered as a rotation around helical axis and the size of this rotation is used to measure the difference (decomposition error);

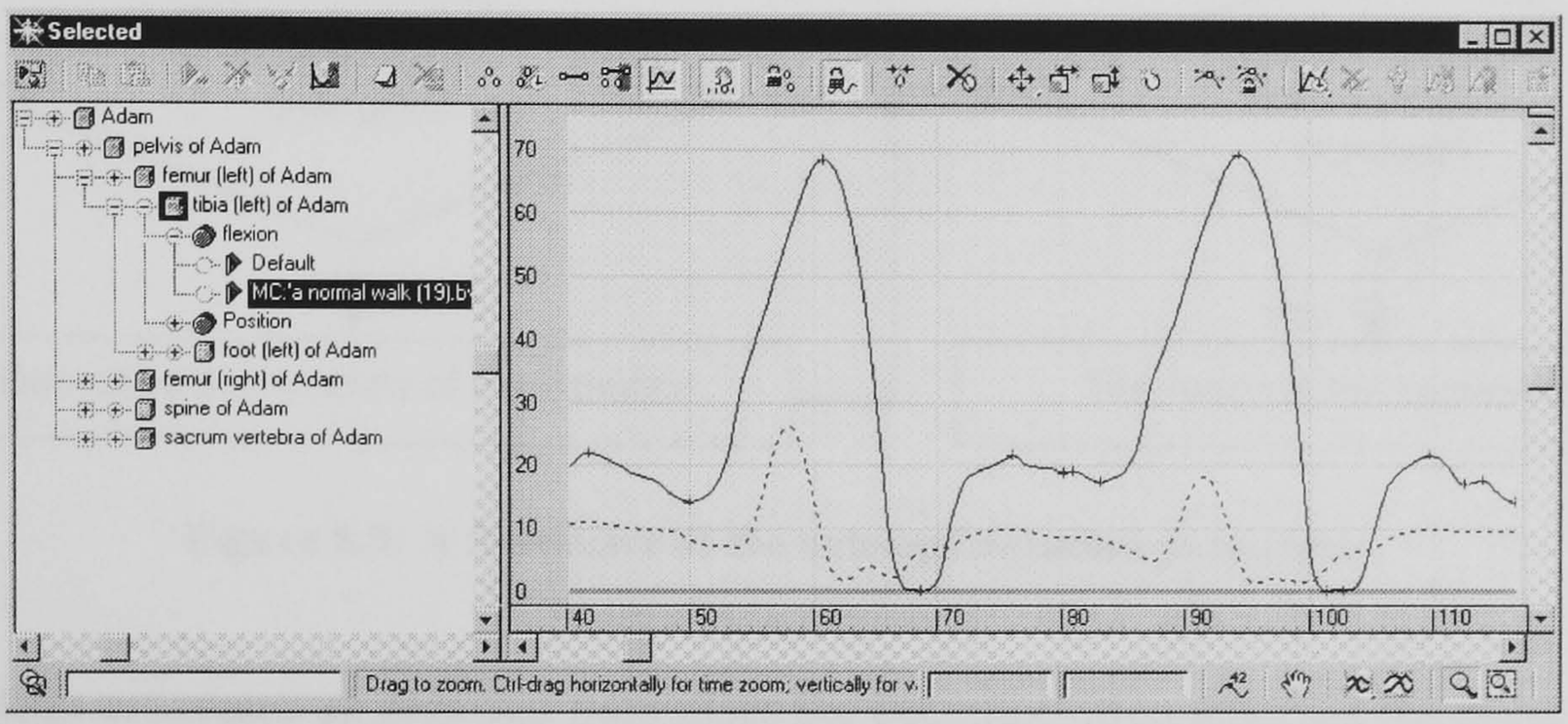


Figure 5.4: The knee-flexion curve (solid) and the corresponded decomposition-error curve (dotted). In this example the knee axis coicide with the horizontal axis (X) of the figure.

Note for joint-motion graphs in this chapter: vertical scale is in degrees, horizontal (time) scale is in frames.

- The original matrix data should not be discarded but should be kept together with the DOF data. First, this gives us an opportunity to adjust joint parameters and quickly update the motion in accordance with these changes. Second, using the original data in Forward Kinematics (i.e. calculating end-effector positions) produces more accurate results since the calculations are not affected by the decomposition error.

Dealing with decomposition error

There are several ways to improve the data accuracy affected by the decomposition error. First of all, one can use a lossless data format i.e. storing joint motion in a quaternion form rather than fixed-axis form. This method however is not supported by the figure model (Section 4.5.1). The alternative solutions are optimising the joint parameters (orientations of the axes) or redistributing the residual transformations to the neighbouring joints.

The idea of the first method is to find an orientation of the joint axis that will minimise the decomposition error over the motion. This can be done using an iterative optimisation technique, whose flowchart is depicted in Figure 5.5.

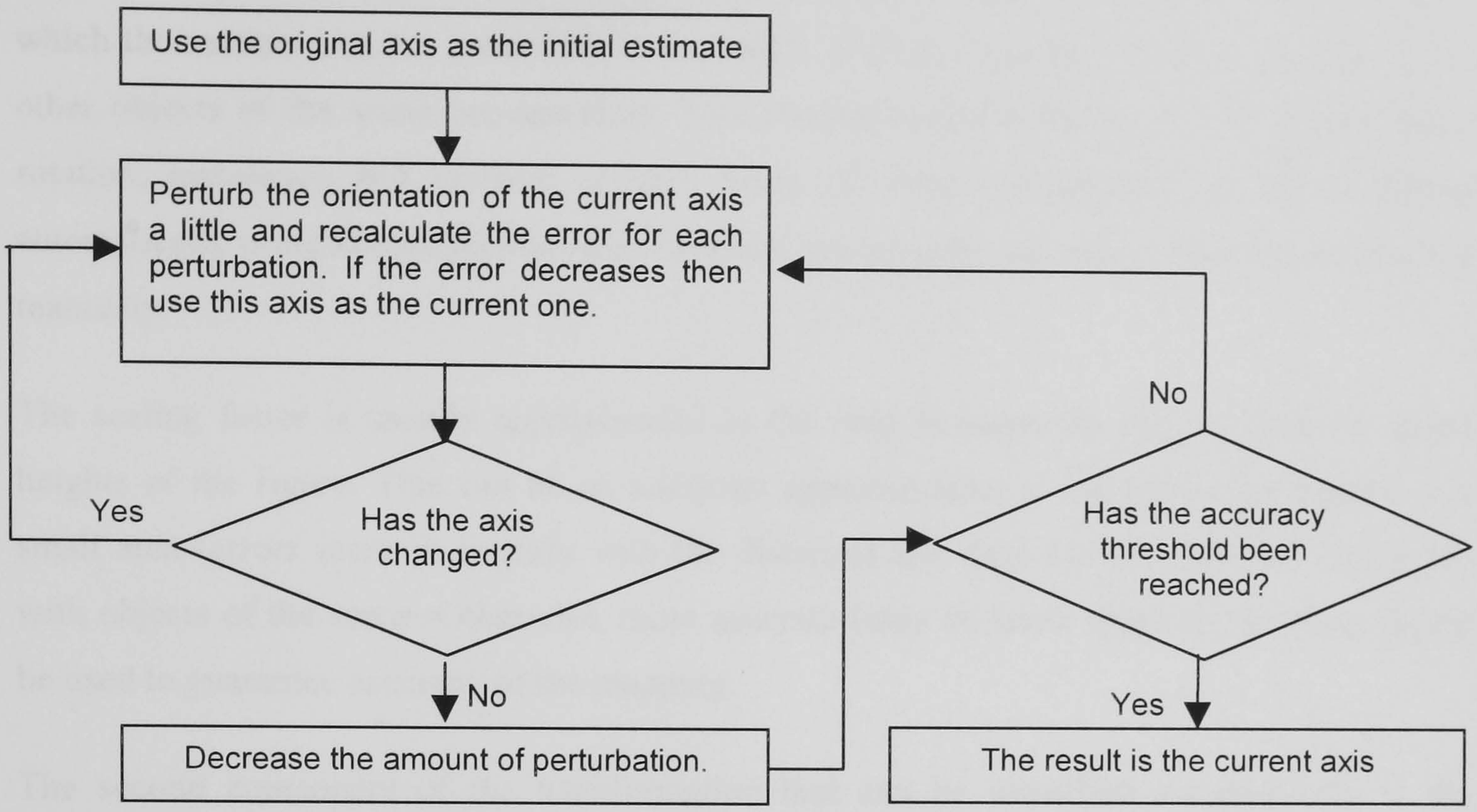


Figure 5.5: A flowchart of the axis-optimisation procedure

This algorithm calculates an axis that minimises the decomposition error for the given motion. However, experiments have shown that the optimal axis does not necessarily coincide with what knowledge of joint anatomy would suggest. Partially, this can be explained by the measurement error (due to the meter error, skin slipping, etc). But the main reason for such variance seems to lie in the algorithms used to map the marker transformations to the joint rotations. These algorithms may put their own restrictions to the joint motion, distribute the motion between the joints in different ways, etc. As the result, the recorded data do not truly reflect the anatomical joint motion.

The second method is prone to the modifications introduced by the mapping algorithm. Its idea is to redistribute the residual rotation, which cannot be represented by the joint, to the distal and proximal joints. For instance, knee rotation and abduction will be redistributed to the ankle and the hip.

5.1.4 But what about the Environment?

After completing all the steps of the importing procedure, we can finally replay the motion. But as we do so, we may see that the figure's motion does not blend with the scene environment: the figure walks on air, goes through objects or tries to reach objects that are not

there. This happens because the importing algorithm does not know how to map the scene of the performance to the computer scene.

This mapping is nothing but a transformation that changes the original reference system, in which the motion data are presented, to the target reference system, in which the figure and other objects of the scene are specified. This transformation consists of three components: rotation, translation and uniform scaling. Some of these components can be calculated automatically using known relationships between two systems; the others have to be specified manually.

The scaling factor is usually approximated as the ratio between the original and the target heights of the figure. This can be an adequate approximation if the motion is limited to a small area (errors increase linearly with the distance) and does not involve any interaction with objects of the scene. Otherwise, more accurate (may be semi- manual) methods should be used to guarantee accuracy of the mapping.

The second component of the transformation that can be identified automatically is the vertical transformation of the figure. To do this, the algorithm first has to identify postures when a figure's foot is in contact with the ground. Then, knowing the target ground level, one can easily calculate the required offset.

Other components of the transformation, horizontal translation and rotation, can be specified manually.

If the data were absolutely accurate and the performer's proportions matched the model's closely, then all post-processing could be finished at this point. However, this is rarely the case. Errors and discrepancies in the proportions of the model may result in various artefacts. The worst of these may require re-recording of the performance or adjusting the scene (for instance, if a virtual football player is missing the ball, it can be either moved or if this is not possible, the performance scene should be adjusted and the motion should be recorded again.). But, for some of these artefacts special post-processing techniques can be used that reduce or eliminate the negative effects. These artefacts include the two most typical problems associated with the use of recorded walking motions:

- *Ground penetration / Flying Foot.*

  The problem of feet penetrating the ground surface or flying above it has two typical causes. The first is the discrepancy between the proportions of the figure and the performer. For instance, longer feet create an inadequate amount of ground-clearance and late push-off (Figure 5.6). The second cause is not accurate reproduction of the target surface-relief during recording.

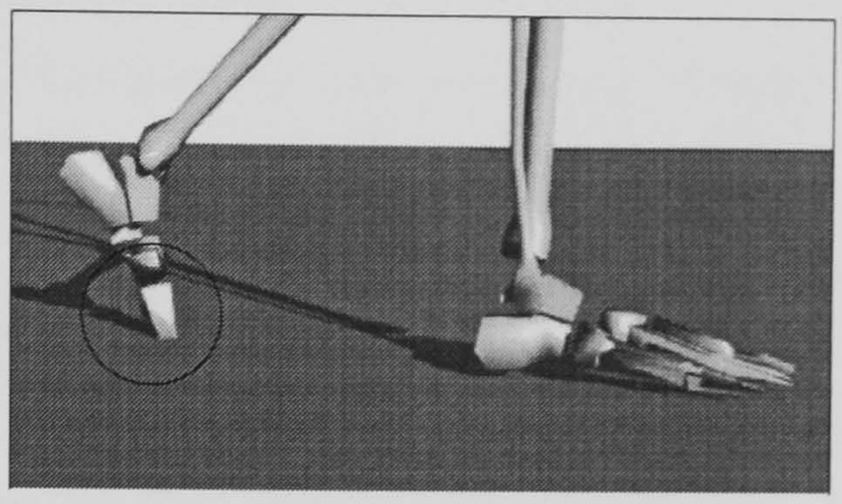


Figure 5.6: Ground penetration

  Depending on the features of the animation (distance to the subject, camera angle, etc) this artefact can become more or less prominent. But if, however, it become noticeable it can seriously spoil the impression from the otherwise-realistic animation.

- *Foot slipping.*

  This artefact appears when the stance foot, which is supposed to be immovable, slips or bounces on the ground (Figure 5.7). This can be observed as an unpleasant trembling of the stance limb and can also decrease the level of realism of the animation. Though this effect may seem to be caused by noise in the data, it is not the case. In fact, the position and orientation of the foot depends upon about 12 degrees of freedom and its motion is a combination of several joint motions working in unison. So if the target skeleton does not accurately reproduce the original one, the joint motions may become dissonant.

Figure 5.7: Foot slipping during *Flat Foot* phase.

A combination of these and other artefacts may results in animations being less realistic. The artefacts can be reduced by using the original skeleton model and minimising errors introduced by transforming the motion data from one format to another. However, this solution could not be used within the scope of the described project since it would eliminate any freedom in choosing the joint models and manipulating their parameters. On the other hand, it would be useless to base any experiments on unrealistic source motions. The solution of this dilemma is to correct the motion.

5.2 Motion Analysis

It is nonsensical to try correct or edit a motion if one does not know the type and the structure of this motion. Of course, it does not present a problem for an animator, who performs the

correction and editing manually. But if the aim is to correct the motion automatically then the program should possess AI functionalities to perform the analysis that the animator does, often subconsciously.

The algorithm introduced in this chapter automates the process of motion analysis. It decomposes the motion sequence into basic parts (strides), classifies the motion, and identifies the constraints needed for its correction. Below is a flow-chart of this algorithm.

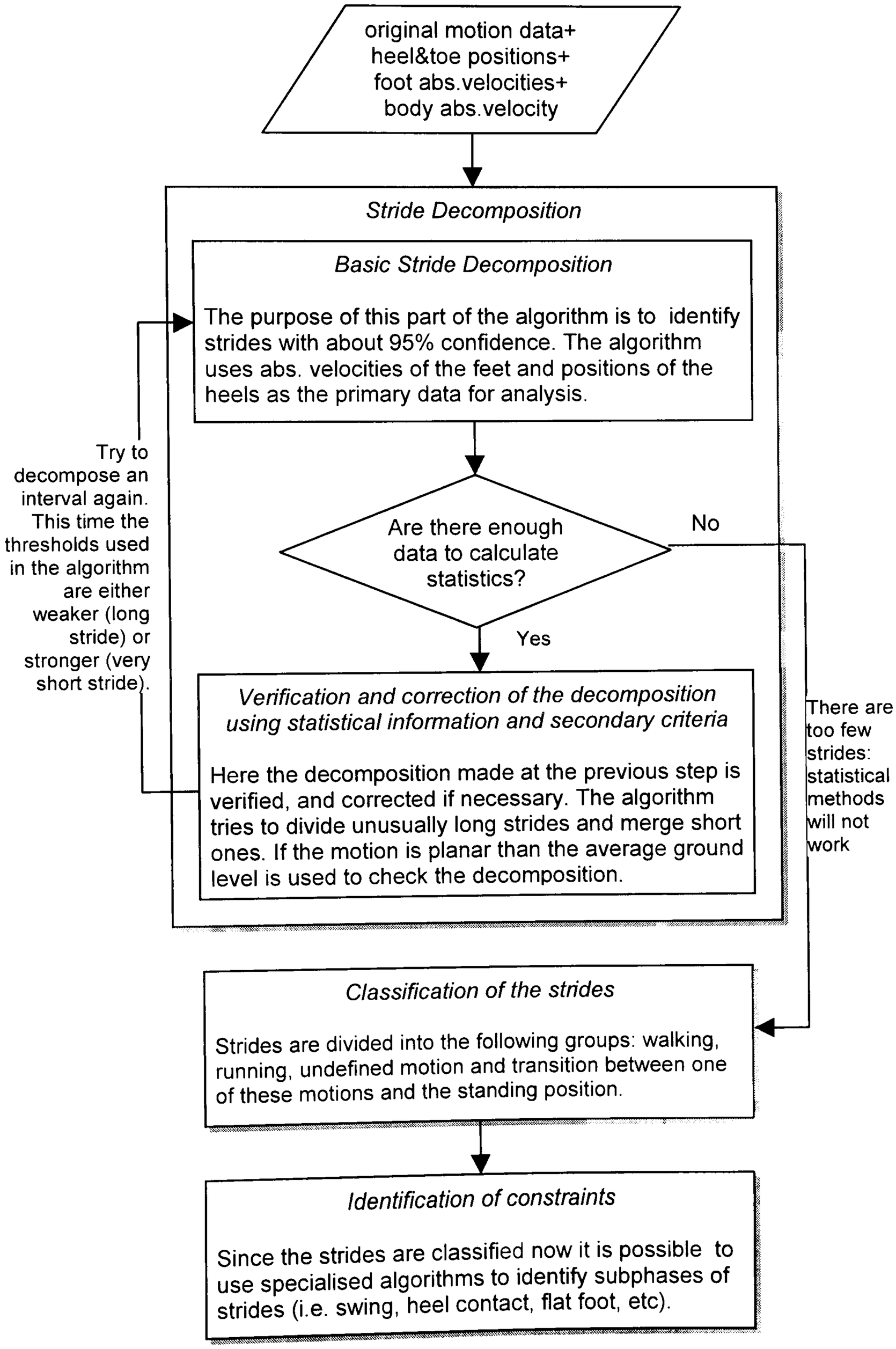


Figure 5.8: Flow-chart of motion analysis algorithm

5.2.1 Data

It was said previously the original motion-capture data consist of joint-rotation data, translational data for the pelvis and optional marker data (translational or both translational and rotational). In addition to these data, the motion analysis algorithms need velocity data for the body and the feet plus positional data for the heels and the toes. If the original data does not include marker data for the heels and toes then these can be calculated using Forward Kinematics. The absolute velocities are calculated as derivatives of the positional curves. Since the original units are usually unknown, the derivatives can be translated into absolute units by normalising them by the figure's height.

5.2.2 Stride Decomposition

The task of this part of motion analysis is to divide the motion into a sequence of strides i.e. intervals between two initial foot-ground contacts of the same extremity. This subdivision serves several purposes: first, it allows different parts of motion to be identified and processed independently. For instance, if the motion sequence includes several activities, such as standing, walking and running, then the algorithm separates walking, running and transitional strides and processes each of them using algorithms adapted for these particular types of locomotion. Secondly, in normal locomotion, strides correspond to gait cycles and the whole motion can be represented as a succession of strides. Therefore, the algorithms can be designed to work with strides rather than with the whole motion sequence; this simplifies the algorithms and improves their reliability (if, at some stage, the algorithm fails to process the motion, it can always resume from the next stride).

The corner-stone of stride decomposition is a determination of the intervals when a foot is in contact with ground. This problem has been addressed both in Biomechanics and Computer Animation, but there are few references in the literature. However, the existing approaches are hardly useful for our purposes: biomechanics people tend to use special equipment (force platforms or special switches) to detect the time and duration of the initial contact with the ground. There have been hardly any attempts to create a general-purpose algorithm using standard MC data only. As for the animators, they usually rely on manual specification rather than automatic methods.

Since neither of these two approaches satisfies the needs of the project, a new method was needed. There were two key-requirements: firstly, the method should be automatic and should require no, or minimum, user intervention; secondly, the method should use only data available in a standard MC file.

Several methods for detecting of foot-ground contact were implemented and evaluated. These methods can be divided into three major groups:

- The idea behind the first group is that the foot (or a part of it) is relatively motionless when it is in contact with ground, so one can look for intervals where a foot does not change its position during some minimum period of time. Obviously, a static position of a foot does not necessarily mean that the foot touches the ground; however the other cases are relatively rare and can be filtered using secondary criteria (these will be discussed later). A more serious problem of this method is how to choose the thresholds for foot position deviation and minimum contact time. There are so many different kinds of motions so it is impossible to select a pair of thresholds that would work in all cases: for example, thresholds that work for a slow motion of an elderly person will not work for running;

  A variation of this criterion is to analyse the absolute velocities of the feet. Though the velocity is more sensitive to noise, the velocity-based threshold can be easily adapted for a particular kind of locomotion and thus the fidelity of the detection can be increased;

- If the ground level is known or can be easily estimated, then it can be used to detect when the feet touch the ground. Though this method can identify ground contacts undetectable by other algorithms, it is too sensitive to noise and error of ground-level estimation to be used on its own. For instance, if the foot clearance during the swing phase is very low – which happens quite often – then the algorithm may falsely assume that the swinging foot hits the ground. As a result of this noise-sensitivity, the method is more appropriate for use as a secondary criterion that is applied when the primary criteria cannot give a definite answer;

- Finally, one can use prominent features of motion curves (joint rotation angles, marker positions, etc) to identify the time of foot-ground contact. Methods based on feature detection are not usually general-purpose methods since they are based on character of a particular kind of locomotion. However, these methods can be very accurate. For example, A. Hreljac [2000] suggests that using the time of maximum of vertical component of heel marker acceleration can be used to approximate the time of heelstrike with an average error of 4ms ($1/10^{th}$ of typical frame rate). This method can be used not only to pinpoint some event, but also to approximate an interval where the event (heelstrike) takes place. Features used for the approximation are usually more versatile than those used for pinpointing and thus they can be used for a wider set of motions.

Experimentation with these criteria demonstrated that none of them is reliable enough to be used by itself across a wide range of motion data. Fortunately, different criteria can supplement each other to make an effective algorithm. The algorithm proposed here follows this idea and it has proved to be highly reliable[1].

In the first stage the algorithm works as a coarse "sieve" that detects intervals that may relate to foot-ground contact. The purpose of this stage is to decrease the amount of calculation by eliminating samples when the foot is certainly not in (active) contact with the ground. The primary data for this and the next stages of the algorithm are the speed of the feet. Since different parts of the foot have different speeds, the overall speed of the feet is calculated as the minimum of the heel and toe speeds and it is normalised by body height.

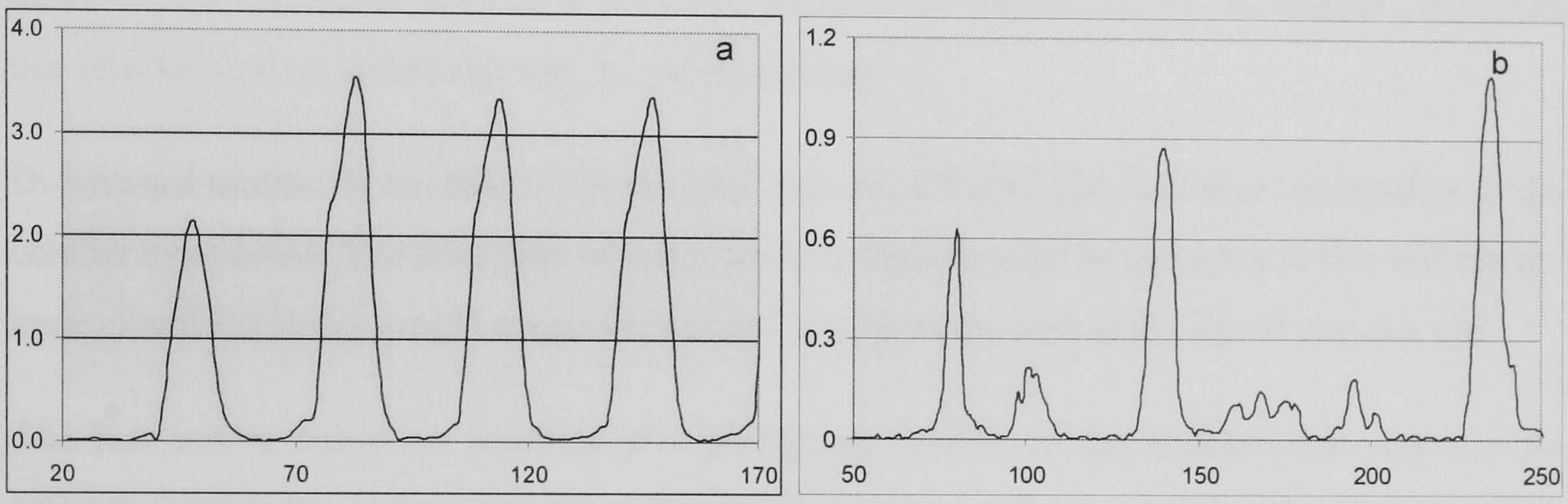


Figure 5.9: Examples of a foot speed curve (vertical scale is in [body heights/per second]; horizontal scale is in frames)

As one can see from the graph above (Figure 5.9a), the periods when the foot is on the ground are characterised by very low velocity. So we can put a threshold, say 0.1, and find all intervals when the velocity is below this threshold. Unfortunately, the data can be noisy and can have different time and amplitude scale (Figure 5.9b). Therefore, initially the algorithm does not use a small threshold to pinpoint every foot-ground contact but uses a higher threshold (0.4) to be sure that it does not miss anything.

In the second stage, the algorithm uses several criteria to check the working hypothesis that the foot is in contact with the ground. The first criterion checks the value of the absolute foot velocity. But in contrast to the initial filtering criterion, this one uses adjusted thresholds and compares them against averaged values of the velocity. The adjustment of threshold values is based on the range of velocities in some neighbourhood of the time of the hypothetical initial

---

[1] The decomposition algorithm was tested on about 50 various motion files. The accuracy of the proposed algorithm was close to 100% for normal walking and running motions, decreasing to 90% for unusual types of locomotion like dancing, elderly walking, etc.

contact: if the body velocity is high, then the thresholds will be slackened; and vice versa, if the velocity is low, the thresholds will be tightened.

The adjusted thresholds are used to estimate the interval of the ground contact. If this interval is too short than the hypothesis is rejected and the search will continue from the next frame. Alternatively the second criterion is applied. It tests that the foot does not move downward near its current position. First, the algorithm approximates the period of time in which the foot stays in a neighbourhood of its current position in XY plane. Then, the minimum of the vertical component of the foot's position in this interval is found and compared to the initial foot position. A significant difference between the minimum and the current Z position means that the foot moves downward, so the assumption about ground contact was wrong. If, however, the difference is close to the threshold then the hypothesis can be neither confirmed nor rejected and an additional test should performed.

Downward motion of the foot is not the only feature of the motion that may contradicts to the contact hypothesis. The examples of other features that can also be included in this test are the knee extension at the weight acceptance phase, rising on the toes at the initial contact, etc.

The last test is based on analysis of a prominent feature of the heel-position curve – its vertical component. If the vertical position of the heel has a distinctive peak (local maximum), then the hypothesis can be accepted.

If the algorithm rejects the hypothesis at some stage, the search should continue from the next frame (or step). However, if ground contact is confirmed, the algorithm should approximate the moment when the foot leaves the ground (*toe-off*) and resume the search from that moment. Though the approximation of *toe-off* using the absolute velocity threshold works well for locomotion-type motions, it can fail in more difficult cases (dancing, sport activities, etc). In such cases, the decomposition algorithm uses secondary criteria (foot position, feature-based criteria) to improve its accuracy and reliability.

Even if the initial stride-decomposition procedure fails, its failures are likely to be detected and corrected during the second stage of the decomposition. In this stage, the strides are verified using statistical criteria: if the duration of a stride is much smaller or larger than an average stride duration (and/or a threshold value specific for the current velocity) then the algorithm repeats the decomposition procedure for the suspicious interval. But this time all the thresholds will be adjusted correspondingly (slacken if a missing stride is suspected or tighten otherwise) and additional criteria (such as ground-level criteria) will be used.

The algorithm described was successfully tested on a wide set of motion data. The success rate for walking and running motions (including transition motions, motions with turning and stopping, unusual and pathological motions) was close to 100%. The rate for other kind of motion was smaller, but still above 90%.

Pinpointing heel-strike and toe-off

The stride-decomposition algorithm above gives an approximation of the *initial contact* and the *toe-off* events. To calculate the exact timing of these events additional heuristics should be applied.

The first approximation of the *initial contact* is chosen in accordance with the velocity-threshold criterion: the instant foot velocity and its average over an interval of predefined length are below a threshold. Subsequently, the algorithm may shift the estimated time of contact to satisfy the secondary criteria. This, however, is done not to pinpoint the time of contact but to ensure validity of the contact hypothesis. Therefore the actual time of the *initial contact* can be earlier or later than the first estimation. A good heuristic for pinpointing the time of the contact is based on the features of the heel and toe velocity curves. As Figure 5.10 shows, the curve of the normalised difference between the heel and toe velocities has distinctive peaks (a minimum in the case of the *heel strike* and a maximum in the case of the toe contact) at the time of the initial contacts. This occurrence can be explained by the fact

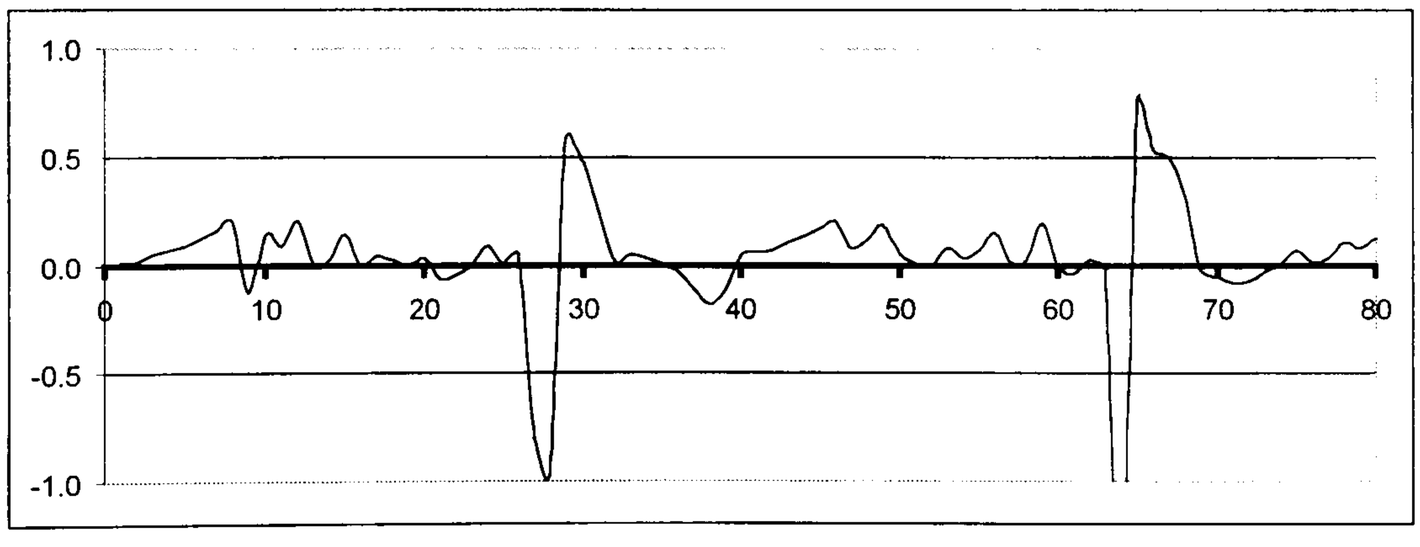


Figure 5.10: Normalised difference between the heel and toe velocities (vertical scale is in [body heights/per second]; horizontal scale is in frames)

that, at the initial contact, the contact point (heel or toe) stops moving while the emergent moment forces the opposite side of the foot to move even faster. This explanation also shows the limitation of the heuristic: it does not work in the case of flat foot contact.

The estimation of the *toe-off* (foot-off, in general) timing needs refining too. However, the accuracy requirements here are less strict. This is because all toe-related data are subject to higher deviations and thus the original phases and events (*the heel-off, the toe-off*) are more likely to change during retargetting.

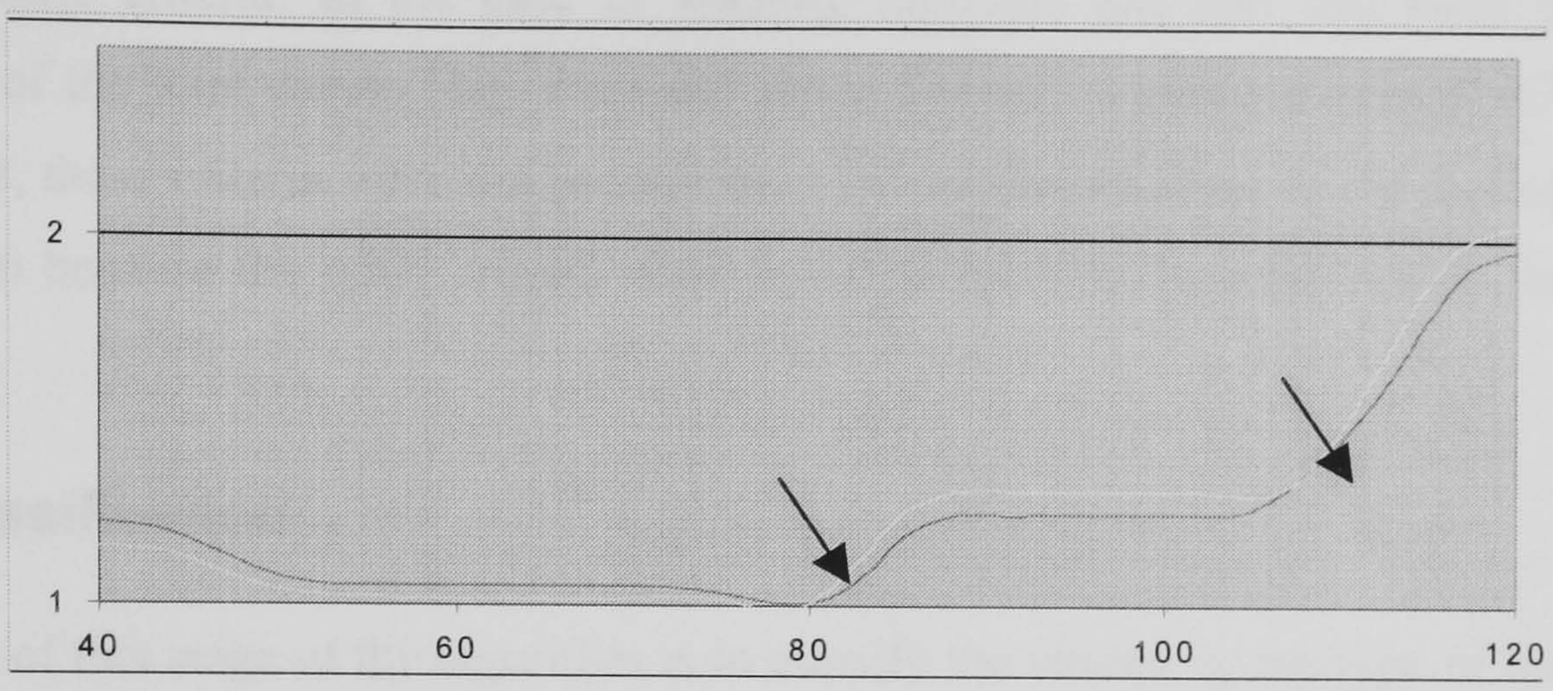


Figure 5.11: Normalised (in 1/170 of body height) position curves for the heel (dotted)and the toe. The "dips" at the ends of the the linear intervals (stance intervals) are due to inaccuracies and absence of the toe-flexion data.

Similarly to the algorithm for pinpointing the *initial contact,* the toe-off algorithm uses the position curves of the toe to pinpoint the time when the foot leaves the ground. These curves show a recognisable pattern (Figure 5.11): a horizontal line followed by an inclined line or curve. The horizontal line means that the toe is static (stance phase), while the inclined line corresponds to the swing phase. These lines intersect at toe-off. If the position data were accurate, the algorithm would be very straightforward. However, many data come in rotational format, so the positional data have to be reconstructed using forward kinematics. Since the accuracy of the rotational data is not high and most of them lack the data for toe-flexion, the calculated positional curves cannot boast accuracy. This means that the algorithm implemented for toe-off should not be sensitive to noise in the data.

Another factor is that the algorithm should not be limited to recognising a typical toe-off only in walking but should be able to deal with most possible motions. To take into consideration all this issues, the algorithm combines several criteria to pinpoint toe-off. These include:

- Distance: the current toe position is compared against the position of the static toe (calculated by averaging toe positions during the middle part of the stance phase);
- Forward Movement: This criterion ensures that the toe advances in roughly the same direction during the initial swing phase. The direction should be close to the foot progression line; if the foot is returning to the same position at the end of the stride (walking on a spot) then all directions are considered equal;

- Curve feature: Toe-off normally coincides with the extrema of the toe-position curve. The most prominent one is the local minimum of the vertical position curve. This particular feature is specific to inaccurate motion data, which ignore toe flexion. Accurate positional data would have a strongly pronounced inflection at the time of the toe-off;
- Gait-specific criteria: in the case of walking motions, one can use some prominent features of the joint curves (hip, knee and ankle flexion) to pinpoint toe-off (Chapter 3). However, these criteria were not used in the final implementation of the motion analysis algorithm because the other criteria were found to produce accurate results for walking motions.

5.2.3 Classification

The purpose of this stage of the algorithm is to classify the strides by the type of locomotion. Such classification allows the correct retargetting algorithm to be chosen for each particular kind of motion. The motions are divided into the following categories:

- **Walking**. Walking is identified here using two criteria: reasonable progression of the body[1] and characteristic alternating motion of two legs (hips). This category is divided into several sub-categories (forward or backward walking, walking on planar or non-planar (staircase, etc) ground, heel or toe walking);

- **Running**. After motion is identified as a walking motion, it is also checked for presence of the *double-support phase*: if the *toe-off* of one side occurs before the *initial contact* on the other side then the motion is re-classified as running;

- **Transitional-motion**. This covers walking and running strides in which a transition to/from standing takes place. From the point view of the animation algorithm, transitional motions between standing and walking or running are considered as walking or running motions not as some third kind of motion;

- **Other motions**. Any other motions that cannot be identified as walking or running motions are put into this "category". These may include both unusual locomotion and non-locomotion motions like dancing, etc. Motions from this category are analysed and processed using general-purpose methods, which do not rely on any information about the nature or character of the motion.

---

[1] This criterion excludes walking on the spot from this category.

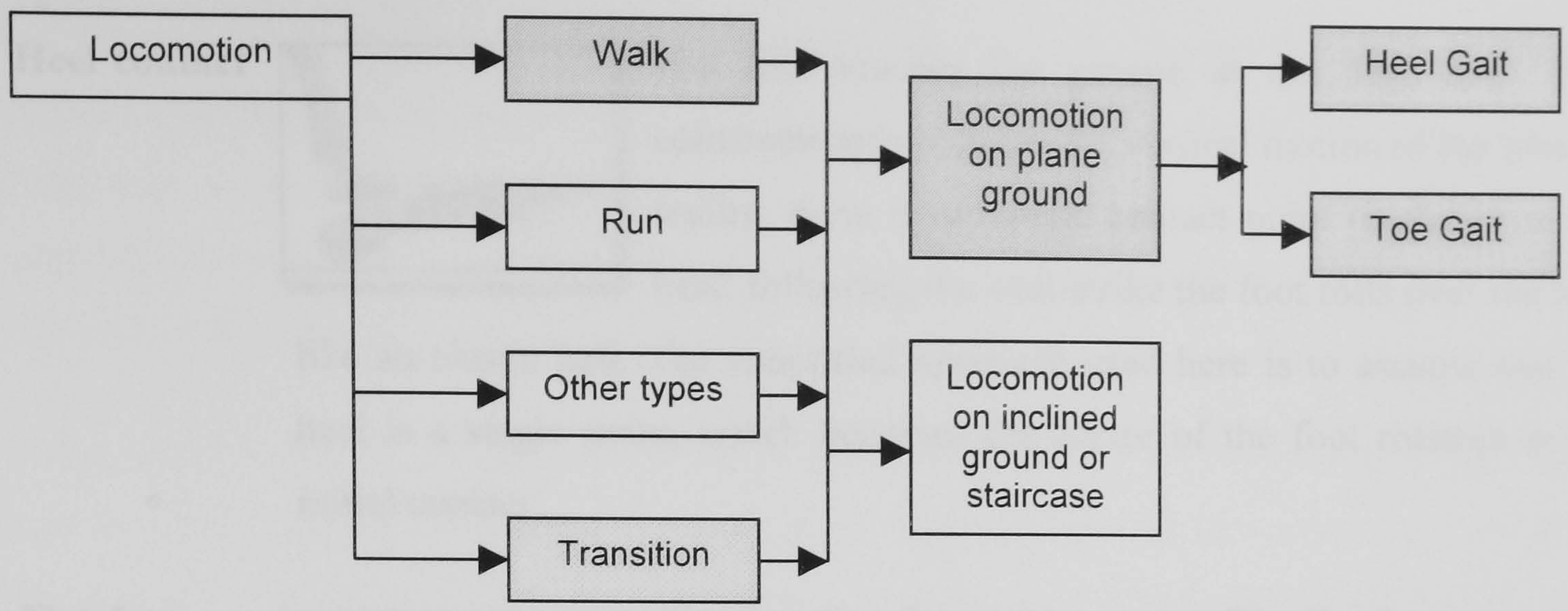


Figure 5.12: Classification of motions used in the model. The selected types are processed using a biomechanics-based algorithm.

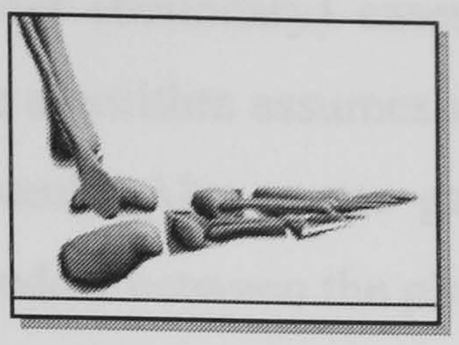
Information about the type of motion is used to perform the detailed analysis of the strides (identify constraints, etc) and then to correct the motion during retargetting. The motions identified as walking on plane ground are processed using a biomechanics-based algorithm adapted for this kind of locomotion (Figure 5.12). Motions of other types are currently processed using a general-purpose algorithm, though design of the system provides for adding other motion-specific algorithms (staircase walking, running, jumping, etc).

5.2.4 Identification of Foot-Ground Constraints

The stride-decomposition algorithm discussed above subdivides the motion into a series of strides that is intervals between two initial contacts of the same foot. It also identifies the moments when the foot leaves the ground, consequently breaking strides into the "contact" and "no-contact" phases (or stance and swing phases in walking terms). This information is crucial for correcting motions since it allows the algorithm to enforce ground contact during the "contact" phase and to check that ground clearance is adequate during the "no-contact" phase.

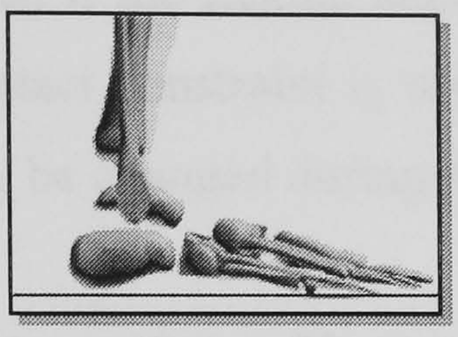
The separation of the motion into just two phases is adequate for stick models, in which a foot is represented by a point. An accurate model, however, requires a more precise definition of "contact" constraints. Our model supports four types of foot-ground constraints:

Heel contact



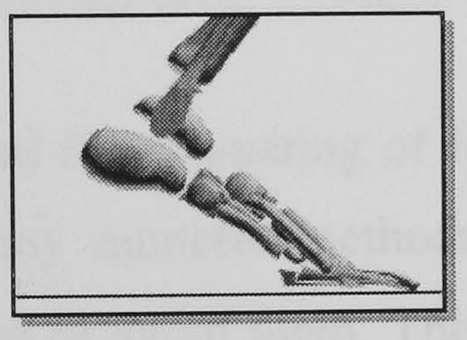
The foot touches the ground at the heel only. This constraint only restricts the vertical motion of the foot. In reality, there is no single contact point (surface) on the heel: following the heel-strike the foot rolls over the heel like an elastic ball. The simplified approach used here is to assume that the heel is a single point, which becomes the centre of the foot rotation at the initial contact.

Flat-foot



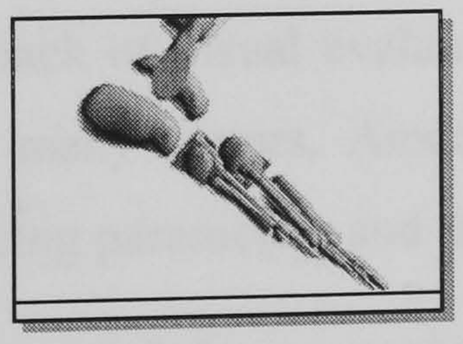
The foot lies flat at the ground. The flat-foot constraint restricts both the vertical position of the foot and its orientation in vertical planes. The second restriction is due to two contact "points": the heel and the metatarsal heads of the foot. As the result of the errors and inaccuracies in calculations of the reference pose, it may be difficult to tell boundary cases of the flat-foot position from the heel or toe contact positions. However, high precision is not required here, since the correction algorithm works within some time interval rather than at a single instance of time; therefore a transition from one constraint to another is smooth.

Toe-contact



The foot contacts the floor with flat toes. Again there are several contact points with the ground and the constraint restricts both the position and the orientation of the toes. In the case of a bare foot, there is a clear distinction between full toe-contact and **toe-end contact** (when metatarsal heads are off the ground). This distinction is less noticeable when shoe-on feet are considered. Nevertheless, it was decided to consider these two kinds of toe contact separately.

Toe-end



The only part of the foot contacting the ground is the toe. Therefore, there are no restrictions on the orientation of the foot. Likewise, there are no restrictions on the state of the metatarsal joint, which can be either flexed or extended.

The easiest way to identify the constraints is to use the foot inclination angle in the sagittal plane. Positive angles correspond to heel contact; small angles correspond to flat-foot contact; negative angles point to toe or toe-end contacts.

To solve difficult (boundary) cases, additional hints are used. If the motion is a walking motion then the algorithm assumes a particular order of the constraints (for example, heel, flat foot, toes, toe-end). Also, some gait-specific features of the joint curves are employed to pinpoint the borders between the phases: the minimum of knee flexion is used to approximate the time of the *heel-off* event.

Since most of motion data do not have toe-flexion data, identifying the last two constraints can be a little difficult. Our approach is to use a maximum toe-flexion threshold to separate these constraints. If the flexion angle needed to keep the toes flat exceeds the threshold then the toe-end contact constraint is used. This separation, though, is not-committal. The actual constraints may be changed during the retargetting.

5.3 Evaluation of the Motions of a Character

While retargetting a motion, the algorithm tries to optimise many aspects of the motion simultaneously. In reality, it is not always possible and one has to choose which characteristics of the motion are more important and should be optimised in the first place. Such a choice can only be made by evaluating and comparing animations produced by different variations of the retargetting algorithm. This section discusses some of the principles of the evaluation that were followed during the development of the project.

The key method for evaluating of figure-animation algorithms is visual assessment. It has a priority over any numeric methods. The method however has some drawbacks and care should be taken to avoid them. The main problem of visual evaluation lies in its subjective nature i.e. different people react differently to the same motions. The subjectivity is less important when the motion is inspected for the presence of visual artefacts, but it becomes a problem when a more subtle inspection is required, for instance when comparing animations produced by slightly different algorithms or by using different joint models.

Another drawback of visual evaluation is that an observer's perception of figure animation depends upon many factors. Among them are the geometry of the figure model, camera settings, rendering parameters and the type of the motion.

- **Geometry model**: In our implementation, the figure model is displayed as a skeleton. A full polygonal model can be used to skin the skeleton model but the complexity of the skinning algorithm limits its use to offline generation of animations. This raises the question whether motion perception depends upon the representation of the figure model. If not we could evaluate the algorithms on simple skeleton models without having to "skin" the skeleton and render the animation offline. But as the practice shows viewer's

perception does depend on the visual representation. This dependency is not straightforward. Some artefacts in motion, identified using the skeleton model, seem to be less noticeable when using a realistic model, while others, which were not apparent in skeleton animations, may emerge. The realism of the model can also affect the ability to detect subtleties in motion. Experiments performed by J. Hodgins [1998] demonstrated correlation between the models and the ability to sense small variations in motions. In particular, for the tested motions, an accurate polygonal model allowed better discrimination of variations in motions than a stick (skeleton) model. All these suggest that visual evaluation should not be limited to skeleton animations only and so all main tests and evaluations were performed using both skeleton and "skinned" models.

- **Camera**: the camera settings have an immediate effect on the perception of character motion. This fact is well known by animators who can hide some motion artefacts by placing the camera appropriately. Our experience has shown that this should not be neglected by developers as well. The effect of such artefacts as ground penetration and foot slipping can be easily overestimated if one observes slowed-down motions rendered with a static orthographic camera. However these artefacts become less prominent when the animation is shown in real-time and with realistic camera positions. On the other hand, defects such as motion jerkiness become even more apparent.
- **Textures, shadows**: it is difficult to evaluate figure motion correctly if there is no visual binding between the ground (environment) and the figure. Such details of the scene as ground surface textures and shadows give us clues about this binding making the evaluation more accurate. For instance, the simple addition of shadows can make a dramatic difference in the perception of the "flying feet" artefact.
- **Complexity and familiarity of the motion**: the simplest and most familiar motions such as walking and running are the most difficult to model because we can easily detect any defects. Therefore, normal walking motions are the most difficult examination for an animation algorithm, such as motion retargetting.

  Since normal walking requires relatively limited joint motion, it cannot fully demonstrate the effects of using various joint models. From this point of view, it is necessary to consider other motions, which may be less usual than normal walking, but which can benefit from the use of an alternative joint model.

When evaluating a corrected animation, one has to pay attention to two issues: first, the character (details) of the motion should not have been changed by the correction; second, the correction should eliminate (reduce) visible artefacts without introducing new ones.

Preservation of the motion details is accomplished visually, by comparing the corrected and the original motions, and numerically, by analysing the difference between the corrected and original motion curves.

The motion artefacts can also be evaluated both visually and numerically. The following table (Table 5.1) describes the most common artefacts and suggests how they can be evaluated:

Artefact	Origin	Evaluation
Jerkiness of the limbs or the body. This is one of the most irritating artefacts. It is particularly noticeable in the motion of the body.	A side-effect of applying discrete numerical methods (Inverse Kinematics) to correct the motion.	This artefact is best evaluated visually, by analysing the animation or the graphs of particular degrees of freedom. Alternatively, if a particular motion can be accurately approximated by a smooth function then the approximation error can be used as the measure.
"Robot-like" motions. There are several reasons of this, including unrealistic accelerations (example: "magnetic shoes" when the foot slaps to the ground too quickly), excessive uniformity of the motion, etc.	Possible origins of these effects include the overuse of interpolation, absence of smooth transition between motions generated using different techniques (for instance, the transition between constrained and unconstrained motion), etc.	"Robot-like" effects can only be evaluated visually.
Collision of the foot with the ground or the "flying" feet Foot sliding during the stance phase	Differences between the original and the target models (proportions, joint parameters, etc) and measurement errors.	Deviation of the foot from the correct position: $\sum_{i=1}^{n} \left\| P_i - P^{correct} \right\|^2 \Big/ n$, where $P^{correct}$ is the correct (expected) position of the foot and $\{P_i\}$ are the actual positions.
Problems with balance.	This effect is often caused by incorrect estimation of the base pose.	The figure model does not provide enough data (mass distribution) for accurate balance control. Therefore this control should be performed visually.

Table 5.1: Artefacts of MC-based animations

5.4 Summary

This chapter provided the background of MC technology and explained the necessity of using Motion Retargetting to correct captured motions. It was also shown how different motion analysis techniques can be used to extract from motion data all the information needed for the retargetting algorithm.

The next chapter, titled "Motion Correction", describes the workflow of the retargetting algorithm that was developed in scope of the project and demonstrates how the information provided by motion analysis is used to enhance the quality and universality of this algorithm.

6 Motion Correction

The aim of motion correction is to adapt the captured motion to a model and an environment different from the originals. This process is limited by two conflicting requirements. On the one hand, the motion correction (retargetting) algorithm has to correct various artefacts that reduce the realism of the animation, but on the other hand, the algorithm has to minimise any changes to the motion in order to preserve its character and details. This conflict represents the main problem for developing a retargetting algorithm.

Experiments and analysis of existing retargetting methods suggested that the compromise can be achieved if the correction algorithm satisfies several conditions:

- The algorithm should prefer global corrections to the local ones. This should minimise the jerkiness caused by discontinuities in the joint motion. It is also important that the algorithm does not address the jerkiness problem by smoothing the resulting motion as this may eliminate important details;
- The correction technique should guarantee that the applied correction is minimal. It is particularly important to preserve the character and smoothness of the body motion as any changes to this motion can be very noticeable;
- The algorithm should preserve the features of the joint motions that are inherent for that motion. For instance, if the motion is monotonic on some interval, it should stay so after the correction. Or if the joint curve has a prominent feature such a local extremum, and this feature can be found in any motion of that kind, then the correction algorithm should avoid shifting it in time unless this change is correlated with changes in other joint motions.

To satisfy these conditions, the proposed algorithm has a layered organisation. The most harmless corrections are performed in the upper layers, while corrections that can seriously modify the motion or have negative side-effects are performed in the lower layers. This ensures that serious modifications are applied only if the motion cannot be corrected by other means. Also, since "harmless" corrections are applied first, they reduce the amount of correction that will be applied by the bottom-layer algorithm, so the side-effects are likely to be reduced, too. The diagram below gives an overview of the proposed algorithm:

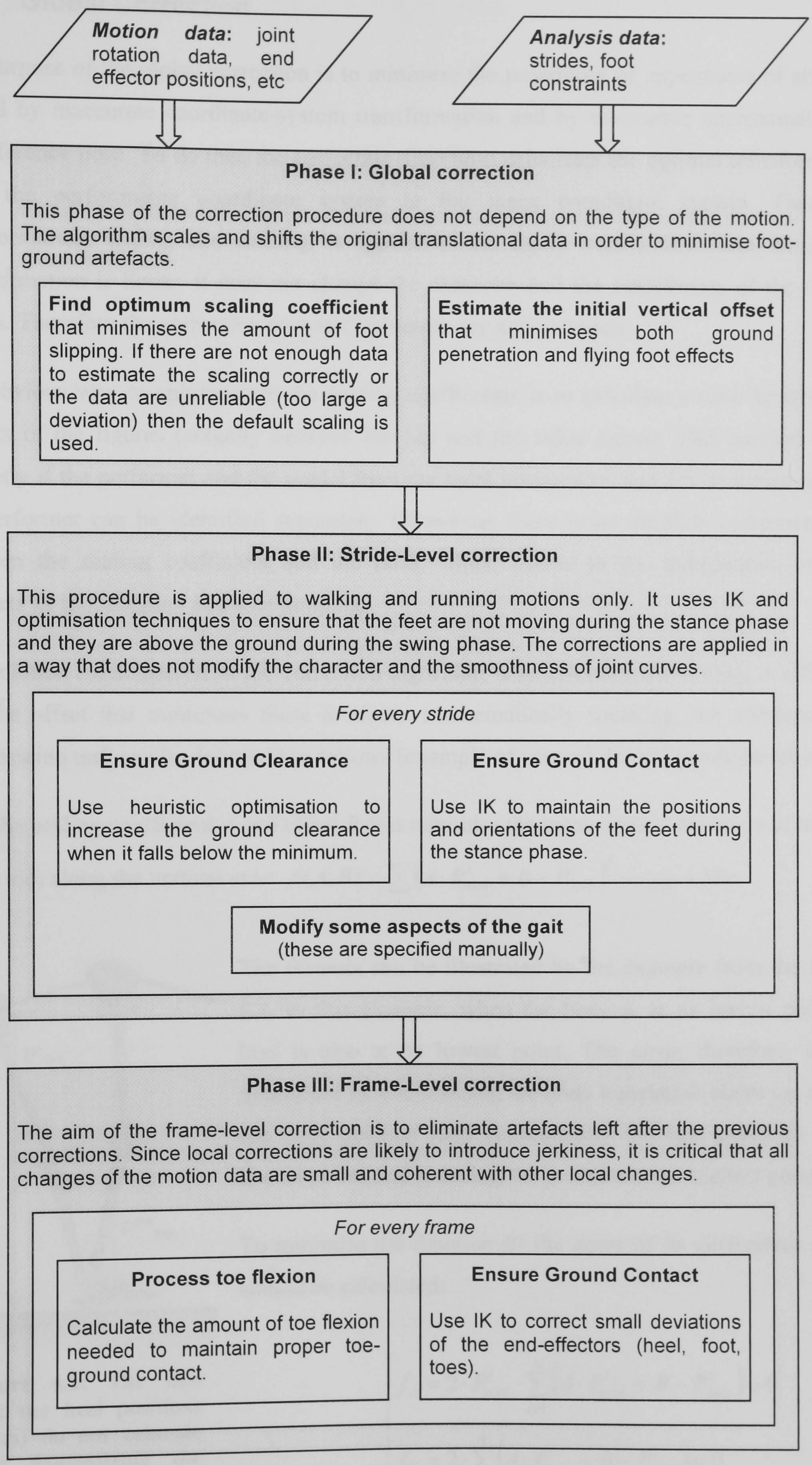


Figure 6.1: An overview of the motion correction algorithm

6.1.1 Global Correction

The purpose of the global correction is to minimise the possibility of appearance of artefacts caused by inaccurate coordinate-system transformation and by inaccurate approximation of the reference pose. To do this, the correction algorithm calculates the optimal transformation from the performance coordinate system to the scene coordinate system. Then this transformation, scaling and shifting, is applied to the figure translational data. Since this transformation is linear, it does not change the character and the smoothness of the motion curves. Therefore the correction does not introduce any new artefacts.

The obvious way to approximate the scaling coefficients is to calculate a ratio between the heights of the figures (actually between the hip and the ankle joints). This method works perfectly if the performer and the model have the same proportions and the reference pose of the performer can be identified accurately. Otherwise, there is no absolute correspondence between the scaling coefficient and the ratio, which results in the introduction of such artefacts as foot slipping and foot jumping.

The solution implemented in the correction algorithm is to calculate the scaling coefficients and the offset that minimises these artefacts. Mathematically speaking, the corresponding optimisation task can be described as follows (example of vertical deviation minimisation).

Find the scaling coefficient A and offset B that minimise the mean-square deviation of the foot (heel, toe) along the vertical axis: $f(A,B) = \sum_i \left(A \cdot P^i_{root} + B - P^i_{foot}\right)^2 \xrightarrow[A,B]{} Min$

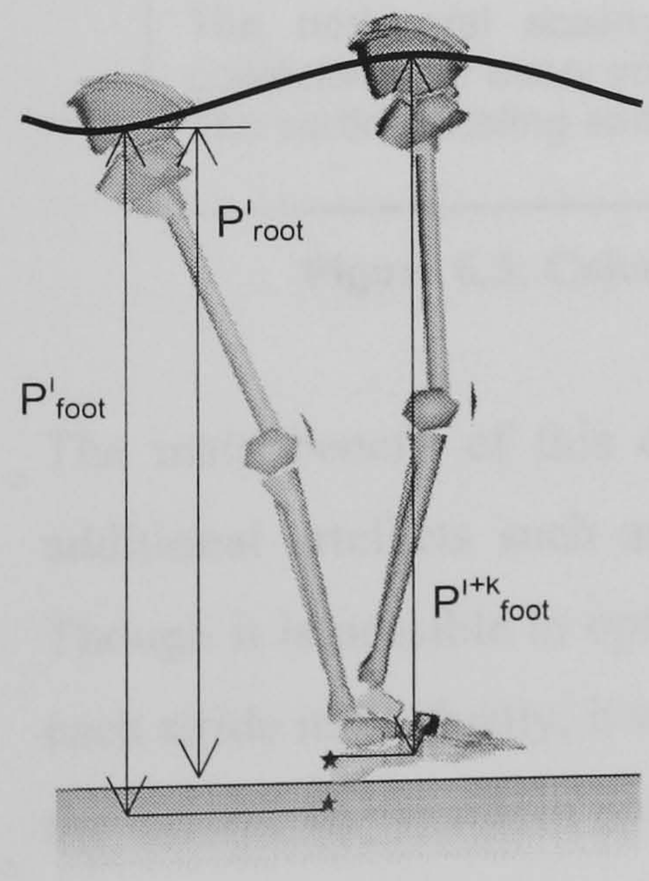


Figure 6.2: The fact that the heel positions (stars) do not coincide here demonstrate the foot-jumping artefact.

The formula can be illustrated by the example from the Figure 6.2. In this example, when the body is at its lowest point the heel is also at its lowest point. The error, therefore, can be minimised by compressing the body translation curve i.e. raising the body position (and consequently the heel position) at the lowest points and reducing the position at the highest points.

To minimise the function *f()* the zeros of its *derivatives* (f_A, f_B) should be calculated:

$$\begin{cases} f_A = 2 \cdot P^i_{root} \cdot \sum_{i=1}^{n} \left(A \cdot P^i_{root} + B - P^i_{foot}\right) = 0 \\ f_B = 2 \cdot \sum_{i=1}^{n} \left(A \cdot P^i_{root} + B - P^i_{foot}\right) = 0 \end{cases}$$

Solving this set of equations gives the following formulas for calculating A and B :

$$A = \left(\sum_{i=1}^{n} P_{root}^{i} \cdot \sum_{i=1}^{n} P_{foot}^{i} - n \cdot \sum_{i=1}^{n} P_{root}^{i} \cdot P_{foot}^{i} \right) \Bigg/ \left(n \cdot \sum_{i=1}^{n} \left(P_{root}^{i} \right)^2 - \left(\sum_{i=1}^{n} P_{root}^{i} \right)^2 \right)$$

$$B = -\left(\sum_{i=1}^{n} P_{foot}^{i} + A \cdot \sum_{i=1}^{n} P_{root}^{i} \right) \Big/ n$$

The full optimisation algorithm is therefore performed in three steps: identify the intervals when the heel (toe) is planted on the ground; calculate the sums required for the above formulae; calculate the coefficients (A and B) and use them to recalculate the translational data.

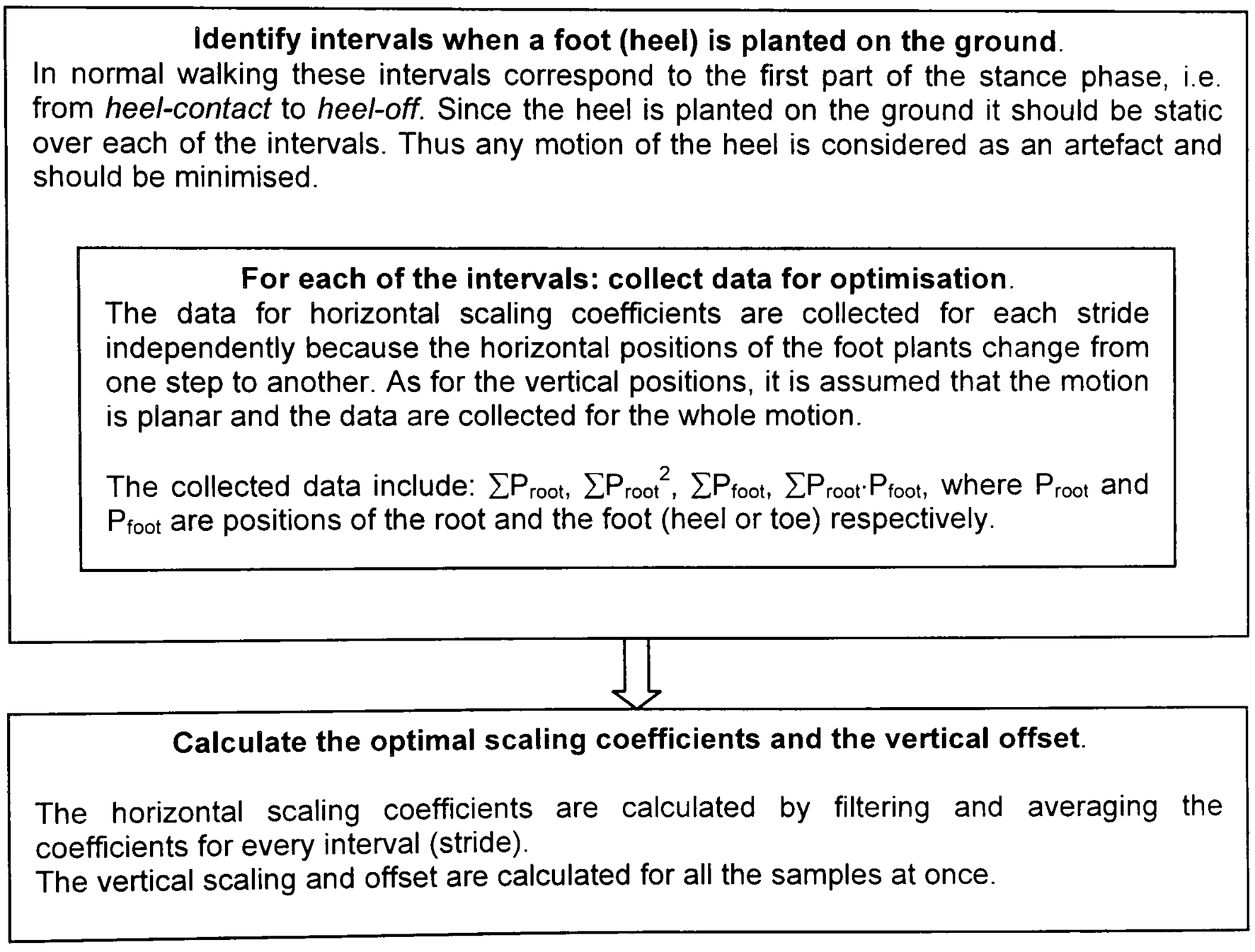


Figure 6.3: Calculating optimal transformation for the translational data

The main benefit of this optimisation is that it improves the motion without introducing additional artefacts such as jerkiness and without changing the character of the motions. Though it is possible to optimise the transformation further by minimising the deviations for each stride individually, it was decided not do so. Using different scaling for different parts of the motion may result in noticeable irregularities in the speed.

Figures 6.4 and 6.5 illustrate the effect of choosing the optimum scaling:

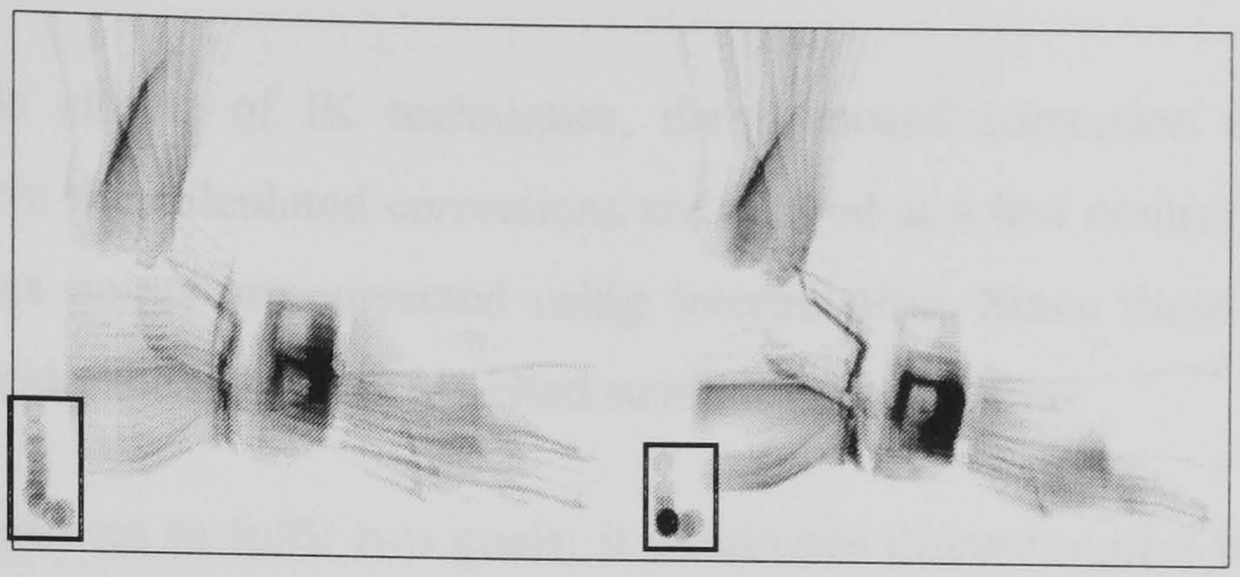


Figure 6.4: (left) With the default vertical scalling; foot is moving upwards following the heel contact; (right) The optimal vertical scaling decreased the deviation

Figure 6.5: (left) Foot slipping with the default horizontal scaling; (right) Foot slipping with the optimal horizontal scaling

6.1.2 Stride-Level Correction

The global-correction algorithm can minimise motion artefacts, but its capabilities are often insufficient to reduce the artefacts to an appropriate level. In order to achieve this level, the correction algorithm has to modify not only the translational data but also the joint angle data. This is a more difficult task because finding joint angles that satisfy positional constraints requires solving the Inverse Kinematics problem.

There are several methods to approach the Inverse Kinematics problem. The most popular of them are based on Jacobian inversion (Appendix A) and non-linear programming techniques. Both methods have been successfully used in the context of figure animation. But though they can completely eliminate the artefacts, there is a pitfall: while the IK algorithm corrects abnormalities in motion of the end-effectors (feet) it may introduce artefacts into the joint motion. The most obvious side effect of the IK-based correction algorithm is that the joint

motion becomes jerky. This effect is a result of kinematic redundancy of the skeleton coupled with the discrete nature of the algorithm.

To avoid the side effects of IK techniques, the proposed correction algorithm uses the following approach: the calculated corrections are applied at a few control points only, while the rest of the data points are corrected using interpolation. Since these control points are unique for each stride, the algorithm is called stride-level correction.

This approach allows us to fulfil two goals: it eliminates discontinuities by interpolating the correction differentials, and it preserves the original character of the motion by using the essential features of the original curves as the control points of the interpolation.

The Figure 6.6 shows a workflow of the correction algorithm:

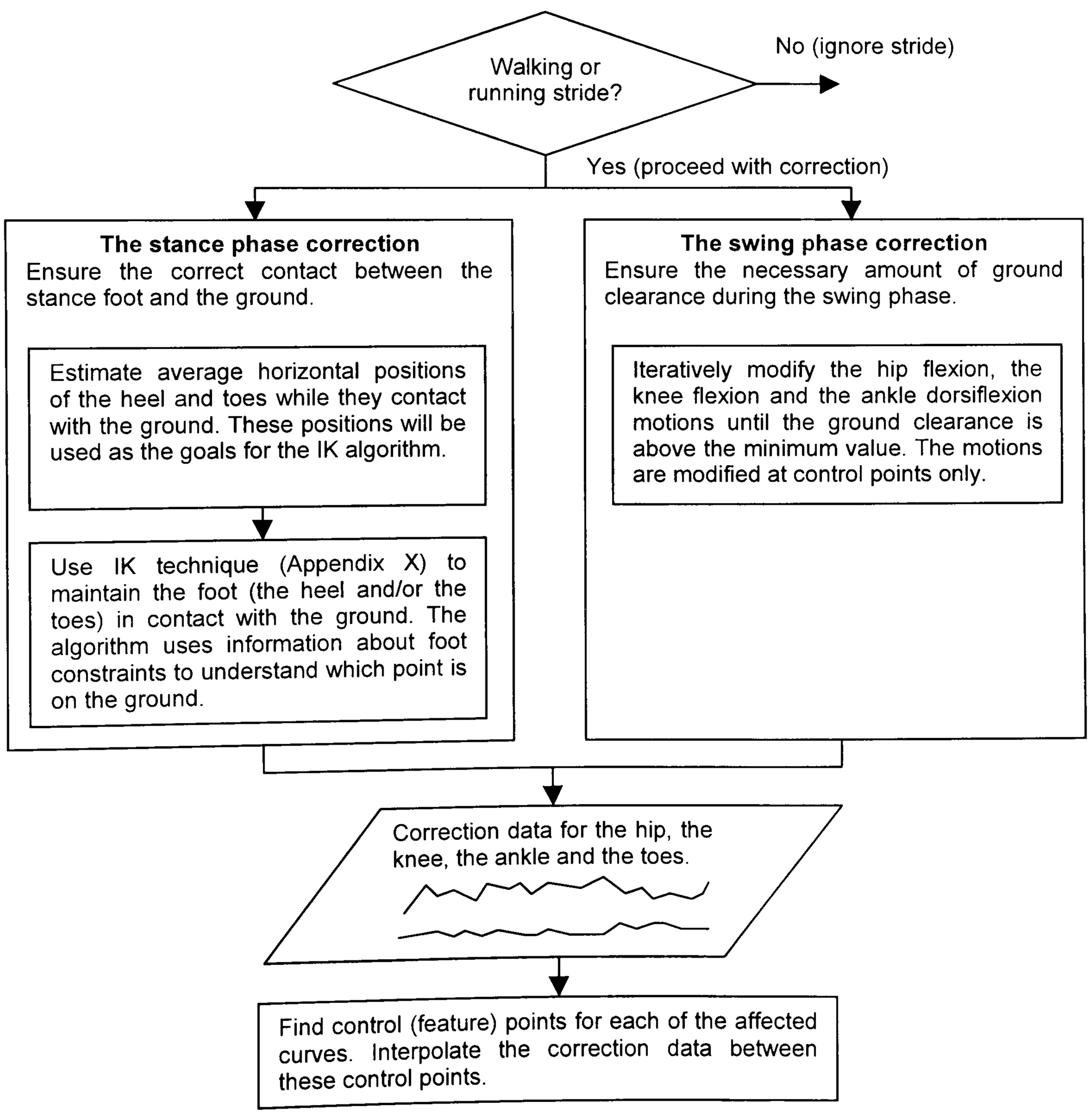


Figure 6.6: A flowchart of the stride-level correction algorithm

As one can see, the algorithm works on a stride-by-stride basis. The algorithm first analyses the type of the stride in order to select the most appropriate method of correction. Currently, only walking motions are supported by the stride-level correction algorithm and, therefore, the strides of the other types will be ignored.

Walking strides are not processed as a whole, but further subdivided into stance and swing phases. Since these phases have different artefacts peculiar to them, they are corrected using slightly different techniques. The stance-phase correction, which takes care of maintaining the contact between the stance foot and the ground, is based on the standard Inverse Jacobian solver. The swing-phase correction, whose aim is to ensure that the swing foot does not go through the ground, uses a heuristic-based IK solver.

The calculated corrections are applied to the motion using a special interpolation technique, which smoothes the effect of jerkiness produced by the IK solvers and preserves the original character of the motion.

Stance-phase correction

The aim here is to ensure the proper contact between the foot and the ground. This means that the contact points (heel or/and toes) of the stance foot should be positioned on the ground, and these points should not move while in contact with the ground.

Mathematically speaking, the task of maintaining ground contact is reduced to solving a system of non-linear equations, each representing a geometrical constraint on the position or orientation of the foot. The free variables of this system are the DOFs of the stance limb, i.e. joint angles. Thus, the solution is a set of joint angles that realises the desired position (orientation) of the foot. To find these angles, the correction algorithm uses an IK solver based on the Inverse Jacobian method – an overview of this method and the IK problem, in general, is given in the Appendix A.

Before the IK solver can be applied to the system of equations, it is necessary to identify the right part of this system i.e. the target positions and orientations of the end-effectors. These are partially derived from the ground constraint: if an end-effector is on the ground then its vertical position should be equal to the current ground level.

As for the horizontal position of the end-effectors, these are defined by the *no-slipping constraint*, which requires that the position of an end-effector must not change until it leaves the ground. This constraint provides freedom in choosing the target position in the horizontal plane, but the optimal position must be decided. The most straightforward way is to take the

position of an end-effector at the moment when it first contacts the ground (*heel contact* for the heel, *flat foot* for the toes). The problem with this approach is that it is sensitive to measuring and motion analysis errors. An alternative, which was used in the algorithm, is to average the positions of the end-effectors over the interval when the corresponding end-effector is on the ground (Figure 6.7). In normal walking, the averaging is performed from the *heel contact* to the *heel-off* and from the *flat foot* to the *toe-off*, for the heel and the toes respectively. Analogously, the averaging technique is used to estimate the target orientation of the foot.

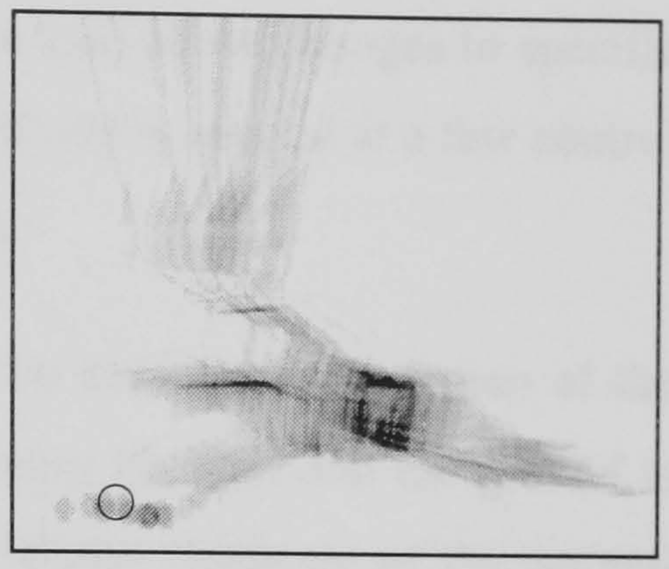


Figure 6.7: Averaging the heel position

Given the initial and target state of the end-effectors, the IK algorithm calculates the joint angles that realise the desired state. It may also happen that the desired position cannot be realised by exploiting the allowed degrees of freedom. This situation may occur due to the error in the motion analysis (omitted toe-off, etc). In this case, one can check the generated constraints for a suspicious stride, correct them and repeat the correction procedure. Otherwise, the correction algorithm will just ignore the stride and resume processing from the next stride.

Swing-phase correction

This is responsible for eliminating another typical artefact of motion retargetting: feet penetrating the ground during the swing phase or, using a more general term, insufficient ground clearance of the swinging foot.

This occurs due to differences in the proportions of the models and measuring/importing errors. To correct the artefact one has to modify the original joint angles and the original position of the body, though the practice has shown that it is better to avoid modifying the positions of the body as it may result in evident artefacts. Therefore, at this stage, the algorithm manipulates the joint angles only.

From the mathematical point of view, the problem of eliminating ground-penetration is similar to the problem considered in the stance-phase correction, and the same technique can be used to solve it. However, that is not the optimal approach. Since there is only one constraint, the system becomes highly redundant. In this situation, it is particularly difficult to achieve a realistic correction using the standard Inverse Jacobian solver.

Instead, the solution used is heuristic. The idea is to take numerous samples of motion-captured data that exhibit the above artefact and correct them manually. This provides

statistics of where the problems usually occur and how to correct them in a realistic way. The statistics are used to adapt the IK solver to correct that particular artefact. In particular, the scaling coefficients for IK DOFs are chosen on the basis of the data about changes to specific joint angles. Also, to minimise the effect of jerkiness, the correction is applied at a few control points only and smoothly interpolated to the rest of the motion.

To correct the motion, it is necessary to have a measure for the numerical estimation of the artefact. The simple measure that calculates the distance between the foot and the ground is not the best variant as it does not take into account the non-linear character of the normal ground clearance curve (Figure 6.8). A good measure should allow different amounts of ground clearance at different parts of the swing phase; for instance, the foot can be close to the ground at the beginning and the end of the phase but should be higher in the middle of the phase. Such a measure is implemented in the correction algorithm using a piecewise linear function, which approximate the average ground-clearance distance.

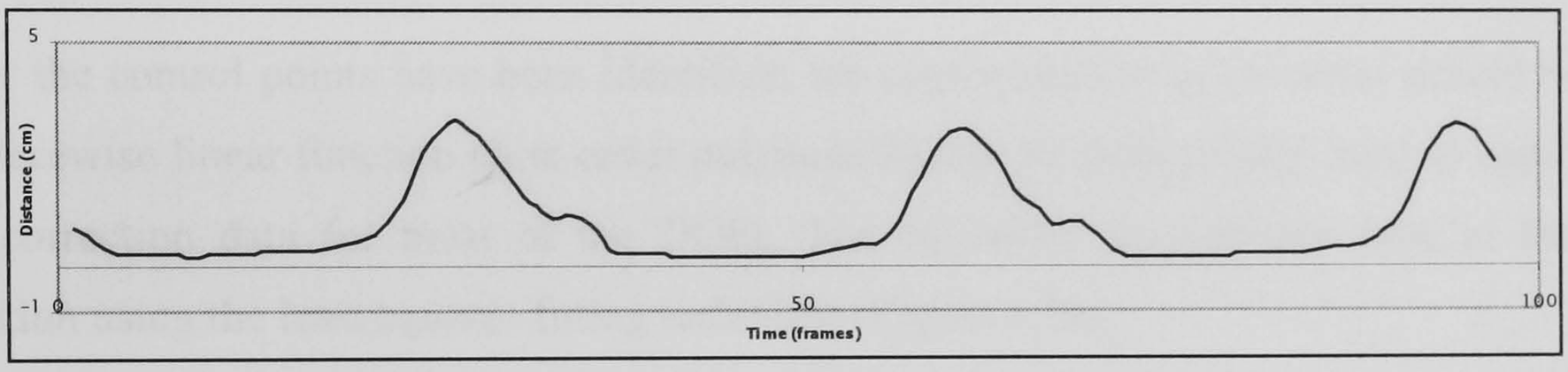


Figure 6.8: An example of ground clearance curve for normal walking (the curve is reconstructed using a Forward Kinematics algorithm)

Applying corrections using an interpolation technique

Since every DOF has its own particular characteristics (smoothness, number of feature points, etc), the calculated correction data are applied to each DOF individually. The way in which the corrections are applied can be illustrated by an example of the knee flexion. The other DOFs are processed in a similar manner.

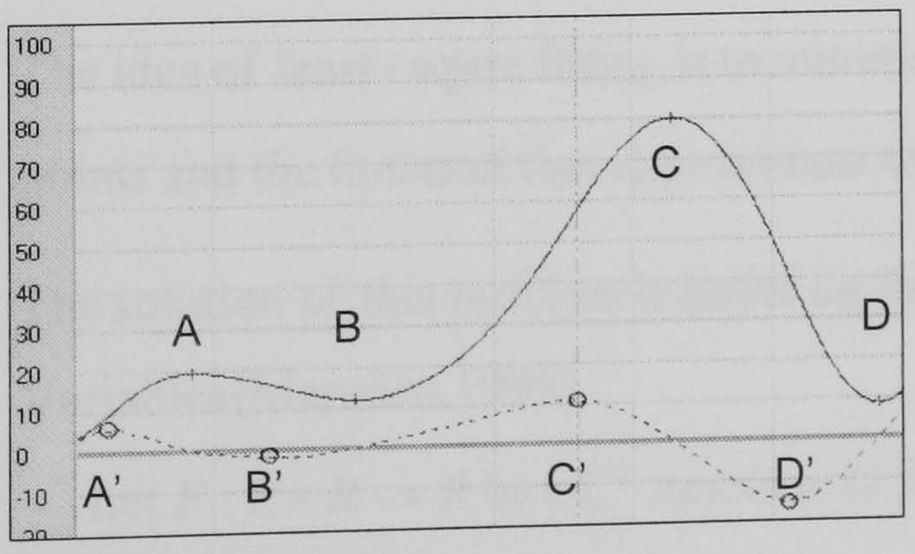


Figure 6.9: The feature points of the knee flexion curve

As the gait analysis shows, knee flexion has a very distinctive and prominent pattern (Section 3.5.2). To preserve this pattern, the algorithm has to add the correction data to the original motion without changing the main features of the curve. In the case of the knee flexion these are monotonicity intervals separated by four critical points: A, B, C and D (Figure 6.9). The monotonicity of the curve can be retained if the correction data are approximated by a

function that is monotonic on the same intervals as the knee flexion curve. An example of such a function is a piecewise linear function with breakpoints at A, B, C and D.

Thus, the process of applying the correction is divided into three phases. First, the features of the curve (critical points) should be located. Second, a polynomial that best approximates the correction data should be calculated. Finally, the approximating polynomial is added to the motion data.

The task of locating the features is not straightforward. Though the general character of the curve remains the same for most of the gaits, there are many small variations (Section 3.5.2) that can complicate the task. The basis idea of the feature-detection algorithm is to identify the most prominent features first and then to use this information and some additional heuristics to locate the other features. In many cases, there is no need to identify the feature precisely: if a feature is indistinct then the approximation error is not likely to be noticeable in the final motion.

After the control points have been identified, the approximation polynomial should be build. A piecewise linear function (first-order polynomial) can be successfully used to approximate the correction data for most of the DOFs. The equations demonstrate how to build this function using the least squares fitting technique (Figure 6.10).

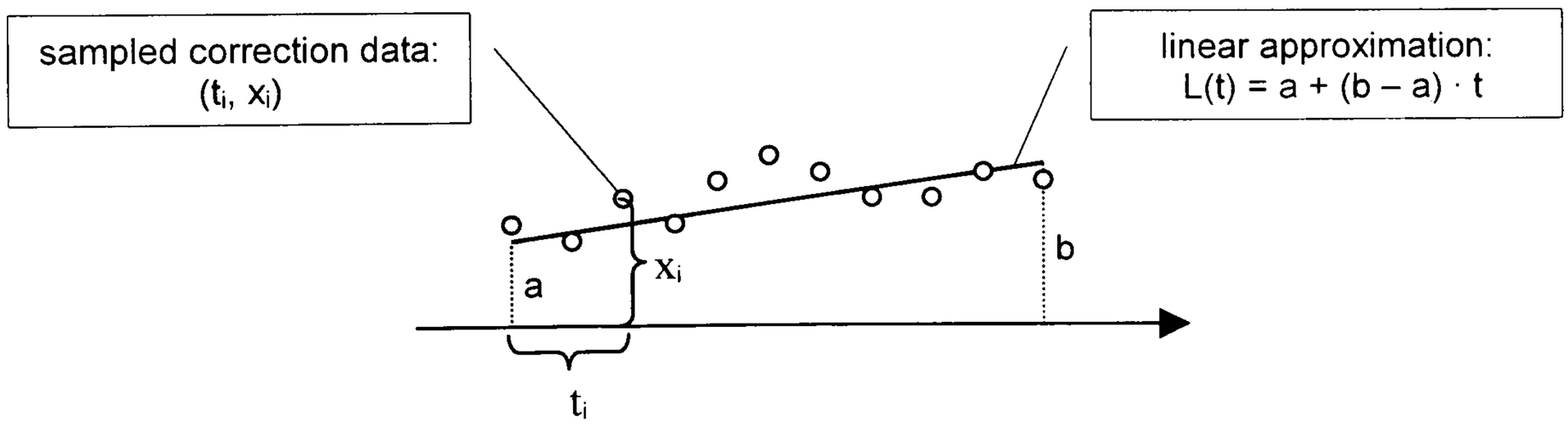


Figure 6.10: Approximating a pointwise function by a first-order polynomial

The idea of least - squre fitting is to minimise the sum of squared distances between the sample points and the function that approximate this points $\sum_i [a \cdot (1 - t_i) + b \cdot t_i - x_i]^2 \xrightarrow[a,b]{} \min$

The solution of this problem is based on the Maximum - Minimum Test for Functions of Two Variables [Marsden, 1996]:

Let $F : R \times R \to R$ be a C^2 function. If (x_0, y_0) is a critical point of f (i.e. $f_x|_{(x_0,y_0)} = f_y|_{(x_0,y_0)} = 0$) and the equation $[(f_{x,y})^2 - f_{x,x} \cdot f_{y,y}]|_{(x_0,y_0)}$ is negative then (x_0, y_0) is an extremum of f.

The differentiation of the minimized function gives the following equations :

$$f_a = 2 \cdot \sum_i [(a \cdot (1 - t_i)^2 + b \cdot t_i \cdot (1 - t_i)) - x_i \cdot (1 - t_i)]$$

$$f_b = 2 \cdot \sum_i [(a \cdot (1 - t_i) \cdot t_i + b \cdot t_i^2) - x_i \cdot t_i]$$

$$f_{a,b} = 2 \cdot \sum_i [(1 - t_i) \cdot t_i], \qquad f_{a,a} = 2 \cdot \sum_i (1 - t_i)^2, \qquad f_{b,b} = 2 \cdot \sum_i t_i^2$$

It is easy to check that *f(a, b)* and its derivatives satisfy the conditions of the theorem and thus the extremum can be derived by solving the system $\begin{cases} f_a = 0 \\ f_b = 0 \end{cases}$. The solution of the system gives the following formulas for the coefficients of the approximating function :

$$a = \frac{\sum_i (1 - t_i) \cdot x_i \cdot \sum_i t_i^2 - \sum_i (1 - t_i) \cdot t_i \cdot \sum_i t_i \cdot x_i}{\sum_i t_i^2 \cdot \sum_i (1 - t_i)^2 - (\sum_i (1 - t_i) \cdot t_i)^2}, \qquad b = \frac{\sum_i t_i \cdot x_i - a \cdot \sum_i (1 - t_i) \cdot t_i}{\sum_i t_i^2}$$

The fact that $f_{a,a} = 2 \cdot \sum_i (1 - t_i)^2 > 0$ guarantees that the extremum is a local minimum.

Using these equations, the approximating functions for each of the intervals are calculated. These gives three pairs of coefficients, one for each internal interval, plus two pairs for the intervals that go beyond the limits of the stride ([D_{i-1}, A_i] and [D_i, A_{i+1}]). To make the polynomial continuous, the coefficients of the neighbouring intervals are averaged (for example, $a`_{BC} = [b_{AB} + a_{BC}] / 2$).

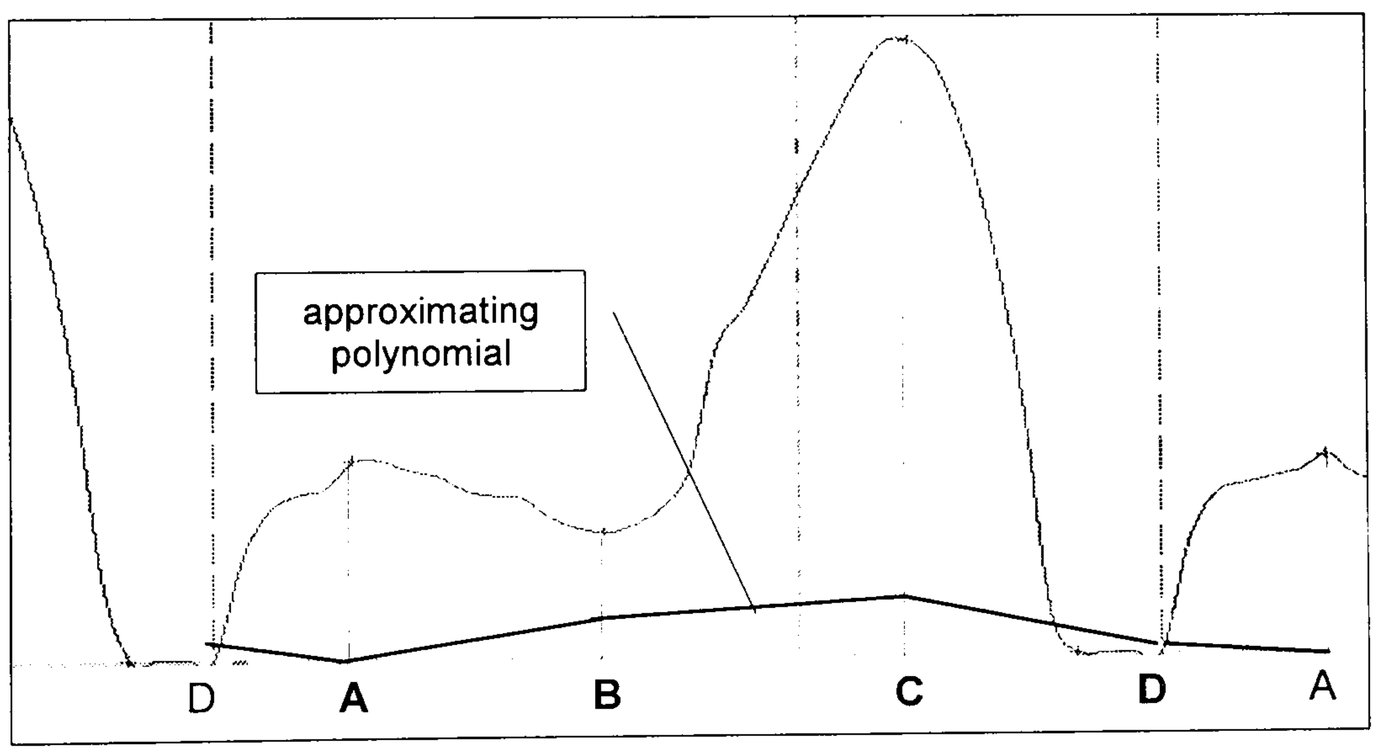


Figure 6.11: Building an approximating polynomial for the correction data

Finally, the motion is obtained by summing the approximating polynomial and the original knee flexion data (represented by a spline).

6.1.3 Frame-Level Correction

The first two phases of the correction algorithm, global and stride-level phases, reduce the artefacts of motion retargetting but do not eliminate them completely. Therefore, the motion should be corrected again to finalise elimination of the artefacts.

This last phase of correction, frame-level correction, works in practically the same way as the stride-level one. The differences appear at the last stage only, when the calculated correction data is applied to the original motion. Unlike the stride-level correction phase, the correction data is applied as it is, i.e. without using interpolation. This guarantees the most accurate motion of the feet, though at the expense of some jerkiness in the motion of the leg.

6.1.4 Test Results

The described correction algorithm was tested on various walking motions, which were captured from different performers and possibly using different MC systems. The evaluation was performed both visually and numerically. The numerical evaluation was based on two numerical measures: one for assessing the stance-phase artefacts and one for assessing the swing-phase artefacts. The formulas for calculating these measures are given below:

$$E_{stance}^{heel} = \frac{\sum_{\text{all strides}} \sqrt{\frac{\sum_{\text{i: heel contacts the ground}} \left\| P_{heel}^{i} - P_{heel}^{average} \right\|^2}{n^2}}}{N_{strides}} \quad \text{(root - mean - square deviation of the heel)}$$

$$E_{stance} = E_{stance}^{heel} + E_{stance}^{toes}$$

$$E_{swing} = \frac{\sum_{\text{all strides}} \left[\sum_{\text{i: gr. clearance constraint is violated}} [G_{clearance}^{i} - \min(P_z^{heel}, P_z^{toes})] \right] \cdot \frac{n_{\text{frames where gr. clearance constraint is violated}}}{n_{\text{frames}}}}{N_{strides}}$$

The first formula (E_{stance}) measures the movements of heel and toe, which take place when they are on the ground. It can be used to estimate foot-jumping artefact i.e. the fact that the feet move when they should be still. The second formula measures (E_{swing}) the violation of the ground clearance constraint and is used as a metric for the foot-ground penetration artefact. Though these metrics do not reflect all aspects of evaluating the motions, they can give some idea how efficient the correction algorithm (or its parts) is.

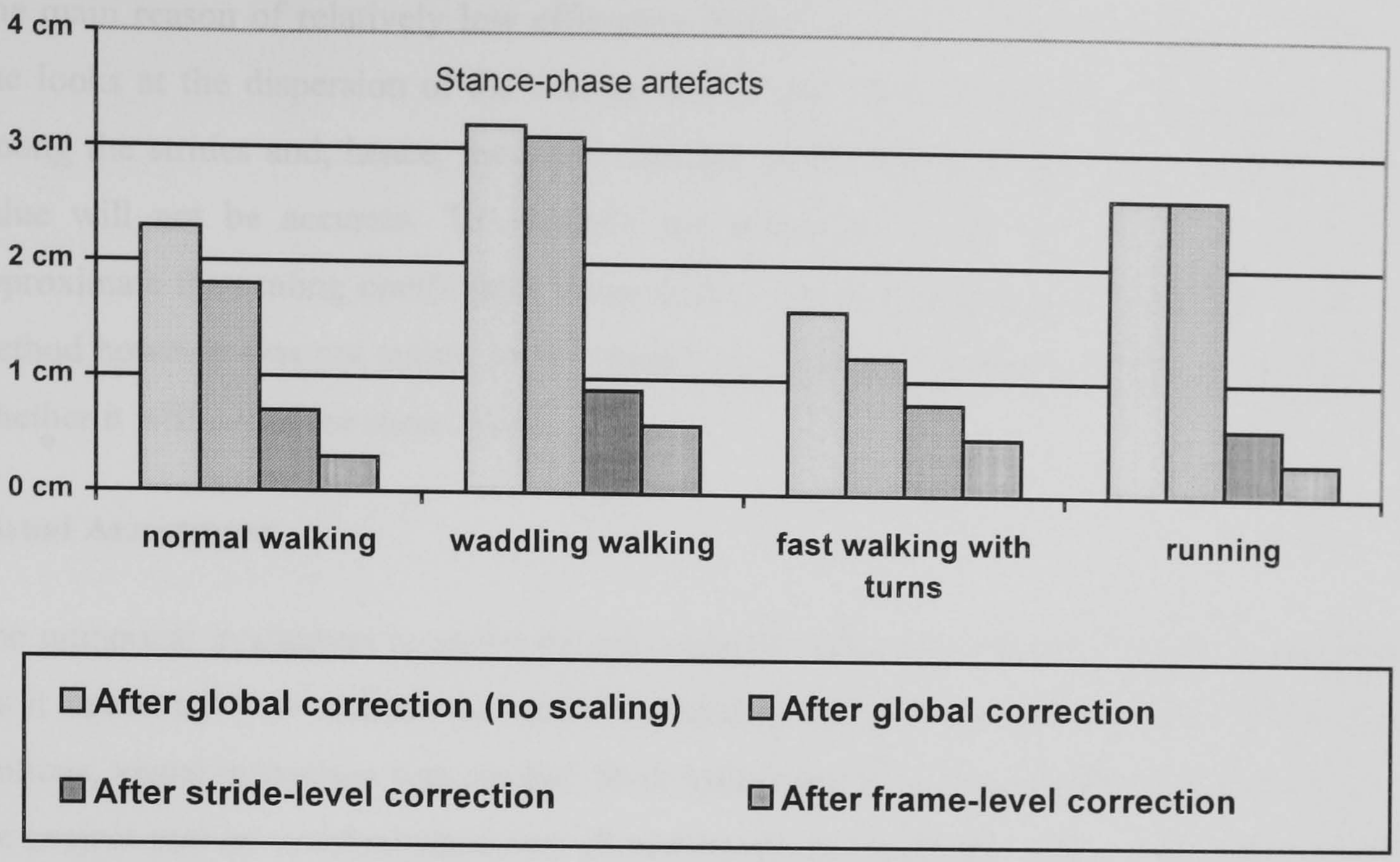


Figure 6.12: A bar chart of the stance-phase error before and after correction

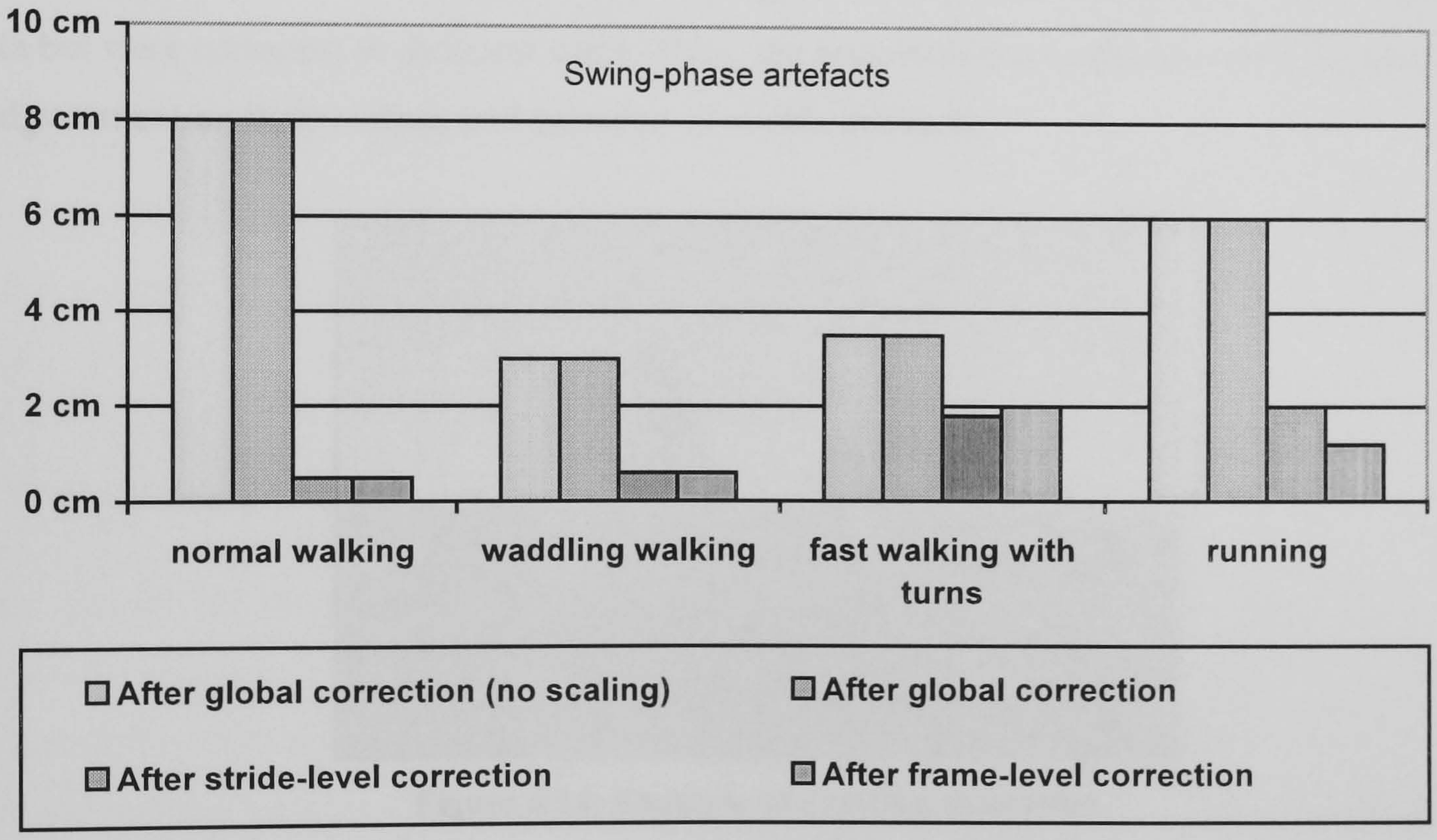


Figure 6.13: A bar chart of the swing-phase error before and after correction

According to Figures 6.12 and 6.13, the first phase of the correction algorithm produces the least effect on the accuracy of the motion. This, however, is not exactly so. The error measure used in the diagrams can be used to gauge the efficiency of one part of the global correction algorithm only; the one, which calculate the optimal scaling coefficients. The other component of the algorithm, which calculates the optimal offset for the translational data, is not taken into account by this measure.

The main reason of relatively low efficiency of the "scaling" correction can be explained if one looks at the dispersion of the scaling coefficients. These coefficients vary significantly among the strides and, hence, the approximation of the scaling coefficients by their average value will not be accurate. To improve the efficiency of the correction one can try to approximate the scaling coefficients using splines rather than using a constant function. This method however was not tested, so it is hard to say whether it will be much more efficient or whether it will introduce some artefacts.

Visual Assessment

The numerical evaluation is useful for assessing the efficiency of the retargetting algorithm, but it cannot tell how realistic the motions produced are. To assess the aesthetic aspects of the motions, visual inspection was needed. Such inspection was performed both by the authors of the project and by external reviewers (8 undergraduate students) using a specially designed questionnaire (Appendix D). First, several animations of figures walking side-by-side were created (Figure 6.14). In each animation, the figures were animated using the same motion data but were corrected in different ways. Then, the reviewers were asked to view the motions and comment on their realism and presence of visible artefacts.

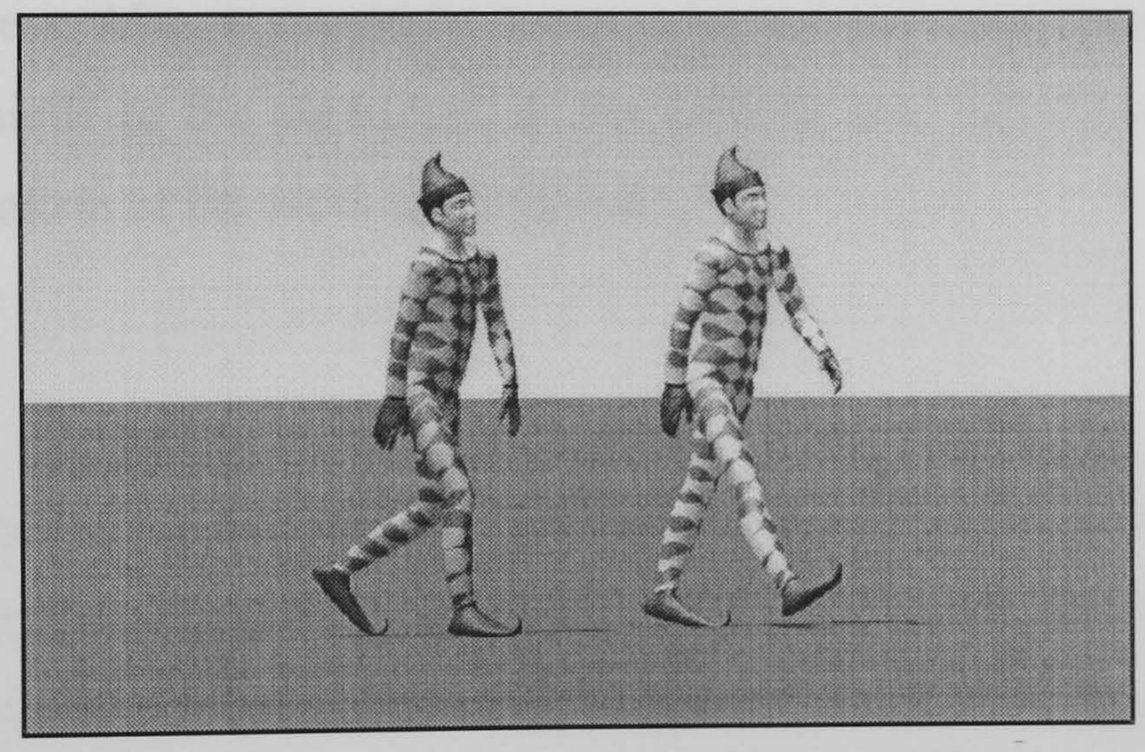


Figure 6.14: Example of a testing animation

The results of this questioning were diverse. People familiar with Computer Graphics tended to pay more attention to foot artefacts, while people who did not know anything about CG left most of these artefacts unnoticed and identified problems in other aspects of the motion, such as figure's balance, movements of the arms, etc. In most cases, the reviewers were unable to notice any difference between the motions after the first or second viewing. Only after analysing the motions more closely could they spot some problems. The most typical of these were jerkiness in the motion of the limbs or body, and incorrect posture ("leans forward too much", "too stiff", etc).

The result of the questioning can be summarised as follows:

- Motions corrected using the retargetting algorithm do not show visible signs of the foot-ground artefacts (flying feet, ground penetration and foot slipping). In some cases, the lowest amount of correction (global correction) was adequate to make these artefacts unnoticeable to all but a scrupulous viewer.
- Upon a more detailed analysis, most of the reviewers noticed that the fully corrected motions are jerkier. Two types of jerkiness were pointed out: limb jerkiness and feet jerkiness. It is interesting to note that the uncorrected motions of the limbs (feet) had many of undesired motions, but these were less noticeable than small, but quick, movements in the corrected motions.
- Motion artefacts such as inadequate balance, stiffness of the spine and arms, etc were not especially considered during the creation of the algorithm but were pointed out as significant factors affecting the total impression from animations.

These results suggest that though the algorithm can successfully correct some motion artefacts, it still has some shortcomings, and they should be corrected before the algorithm would be able to produce motions indistinguishable from the real ones. In particular, future developments should concentrate on improving the Inverse Kinematics engine, which is the main reason of jerkiness. Also, it should be necessary to consider incorporating some kind of balance-control technique in the algorithm, as the balance was the second most noticeable factor affecting the realism of the motions produced.

6.2 Summary

This chapter introduced a novel approach to retargetting human locomotion, in which the traditional general-purpose retargetting techniques are supplemented by methods specifically designed for correcting human locomotion. The proposed retargetting algorithm fully utilise the information provided by the motion analysis algorithm and able to correct motions without modifying their original character and introducing additional artefacts. Together, these algorithms represent a powerful tool for correcting and re-using motion capture data.

7 Evaluation and Concluding Remarks

If one observes the evolution of Character Animation, one will notice that it mainly goes in the direction of increasing complexity of the computational methods. Inverse Kinematics, Dynamics, Motion Capture and even Keyframe Interpolation - all these methods use increasingly complex mathematical apparatus, which are often not correlated to the subject modelled. It is far less common to attempt to question some of the basic assumptions used in Character Animation or to perform a deeper investigation of the basis of the subject, human motion. This is why this research was not aimed at introducing a new computational algorithm but at finding out how the existing animation algorithms and models can benefit from deeper understanding of the human anatomy and the biomechanics of human motion

This chapter summarises the outcomes of this research. The first section discusses the results of incorporating knowledge about anatomy and kinematics of the joints into the figure model. In contrast to Chapter 4, in which the model was introduced, this section surveys it from the position of the quality of animation rather then from the position of biomechanical accuracy. The second section reviews the benefits that MC-based animation techniques can obtain from "knowing" the biomechanical basis of the simulated motion. This review is based on a comparison of existing approaches to motion retargetting and the approach proposed in this project. The thesis concludes with a proposal for possible future work and a summary.

7.1 Evaluation of the Figure Model

The figure model designed and implemented in this project has several novel features that distinguish it from standard models used in CA. The most important of these is the capacity to use biomechanics-based models of joint motion. In this way, the figure model supports different types of joints; it allows joints to have oblique and moving axes of rotation and even takes into account kinematic dependencies between individual joints. These biomechanical features were not added to the model just to make it "biomechanically accurate" but to learn if they can be used to make the figure animation algorithms more versatile and the resulting animations more realistic.

Numerous experiments were made to answer these questions. In the experiments, the figure model was animated using the retargetting algorithm introduced in the previous chapter. This algorithm adapts the source motion to the target model, which allow us to vary practically any aspect of the model (joint model, their parameters, etc) and then compare the animations to analyse the effect produced.

The evident drawback of this approach is its incompleteness. The analysis does not take into account other animation techniques, such as dynamics-based or procedural techniques, that may reveal some other aspects of the studied model. Unfortunately, little can be done about this. As was shown in Chapter 2, the existing animation techniques that are not based on MC do not support the same level of realism as MC-based methods (with the exception of keyframing). These techniques have to improve considerably before they can reach the level where an accurate approach to joint modelling may be significant.

The next few sections analyse the "biomechanical" features of the model. In performing the analysis, these features were examined from three positions. The produced animations were analysed to find any visual changes in motions caused by the studied feature. Then, numerical statistics, such as error measures, were examined to find correlations between the accuracy of joint modelling and the accuracy of the generated motion. Also, such aspects of the model as its complexity and efficiency were evaluated both from a user's and developer's points of view.

7.1.1 Use of Different Models of Joint Motion

One of the most important features of the proposed figure model is that joint motion can be modelled differently in different joints. This approach is much closer to human anatomy than the typical approach, in which all the joints behave in the same way and can only vary in the number of degrees of freedom (DOF).

Real human joints can differ not only in the number of DOF but also in the type and parameters of the motion produced. For instance, uniaxial joints (one DOF) are divided into pivot and hinge joints; biaxial joints into condyloid and saddle joints; multiaxial joints are divided into plane and ball&socket joints. Also, several joints may be so closely tied (structurally and functionally) with each other that they can be only considered as a single unit – a joint complex.

This variety of joint types was taken into account when designing the figure model. The architecture of the model puts almost no limitation on the creation of a motion model for a particular joint: the joint may have any number of DOFs, be kinematically dependent upon other joints, have arbitrarily oriented, or even moving, axes of rotation, etc. This flexibility made it possible to incorporate biomechanical knowledge into the model and to study its effect on the animation of human locomotion.

The foregoing is applicable not just to one or two joints but to practically all joints of the figure model. This is demonstrated in the following table:

Joint	Model
Hip	The hip is the only joint in the human body that can be accurately modelled using the standard robotics model. This, ball&socket model, represents the motion as a series of rotations about three orthogonal axes (X, Y and Z).
Knee	The knee is an extremely complex joint. In fact, it is a multiaxial joint but since two of its motions have a very limited range, it is modelled here as a uniaxial joint. But even this simplified case cannot be handled using the standard robotics model - the axis of knee flexion is inclined and it changes its line of action and orientation throughout the motion. This requires a special joint model that supports an arbitrary oriented and moving axis.
Foot complex (ankle)	The foot complex has two main degrees of freedom, plantarflexion and inversion, both taking place in several anatomic planes. This means that the standard ball&socket model is not applicable here too and a special model, the biaxial joint model, should be used.
Spine	Normally, the spine is modelled as a set of three or four independent joints. This approach however does not take into account the fact the intervertebral joints are kinematically dependent upon each other – such a simplified model may produce anatomically incorrect (and most likely unrealistic) motions. Here, this fact was taken into account creating a model that is both anatomically accurate and efficient.
Joints of the arm	Since these joints were not considered important for the simulation of human locomotion, no specialised models were created for them. However, experiments with various motions demonstrated that none of the created models can reproduce the motions in these joints with adequate accuracy. Both the upper-arm joint complex (shoulder, clavicle) and the lower-arm complex (elbow and wrist) exhibit complex kinematic dependency between compound joints and accurate modelling of these complexes requires the creation of specialised joint-specific models.

Table 7.1: Modelling major joints

7.1.2 Effects of Combined Motion

Combined motion is a term for a biomechanical effect in which the motion of one joint is always accompanied by other motions. The motion created by the external load is called the main motion, and the accompanying motions are called combined or coupled motions.

The effect of combined motion can be observed in many joints and joint complexes, including those that are particularly important for locomotion: the knee, the foot complex and the spine. That is why it was important to support combined motion in the biomechanics-based model.

The simulation of combined motion does not represent any problem: mathematically, this effect is defined by the obliquity of the joint axis[1], which causes the motion to occur in several anatomical planes rather than in a single one. So, to accommodate the effect the joint model was designed to support arbitrarily oriented axes.

To understand how the use of accurate joint axes affects animations of human locomotion, animations produced for two different figure models were compared: one model had robotic-like joints and the other had anatomy-based joints, that were able to simulate the combined motion. Since the models were animated using the same motions, it was possible to elicit the changes caused by using the anatomical joints. This analysis was not only performed visually (by comparing the animations), but also numerically, using the statistics collected during importing and correcting the motions.

The aim of the first part of the analysis was to see if the use of anatomical axes in uniaxial and biaxial joints allows a better approximation of the original joint motion than the standard joints with orthogonal axes. This analysis was performed by comparing the approximating errors (see Section 4.5.3) and by calculating the "optimal" axis that best approximate the given motion. The first experiments showed that some MC data were preprocessed in such a way that secondary components of joint motions were either deleted or redistributed to neighbouring joints. These data were excluded from the test. Analysis of other motions, which

---

[1] Since the axes used in 3-DOF joints are fixed, another approach has to be used to simulate the effect of combined motion. In the spine and neck, the effect is simulated by specifying ratios between joint motions (for example, neck abduction is accompanied by neck rotation, and the ratio between them is 10 to 3).

did not have signs of such processing, showed that the use of anatomical axes[1] does not necessarily result in a decrease of the approximation error (Figure 7.1): in some cases, the error decreased, whereas, in many other cases, the error increased slightly. On the whole, there were no significant differences between the approximating errors of standard and anatomical joints.

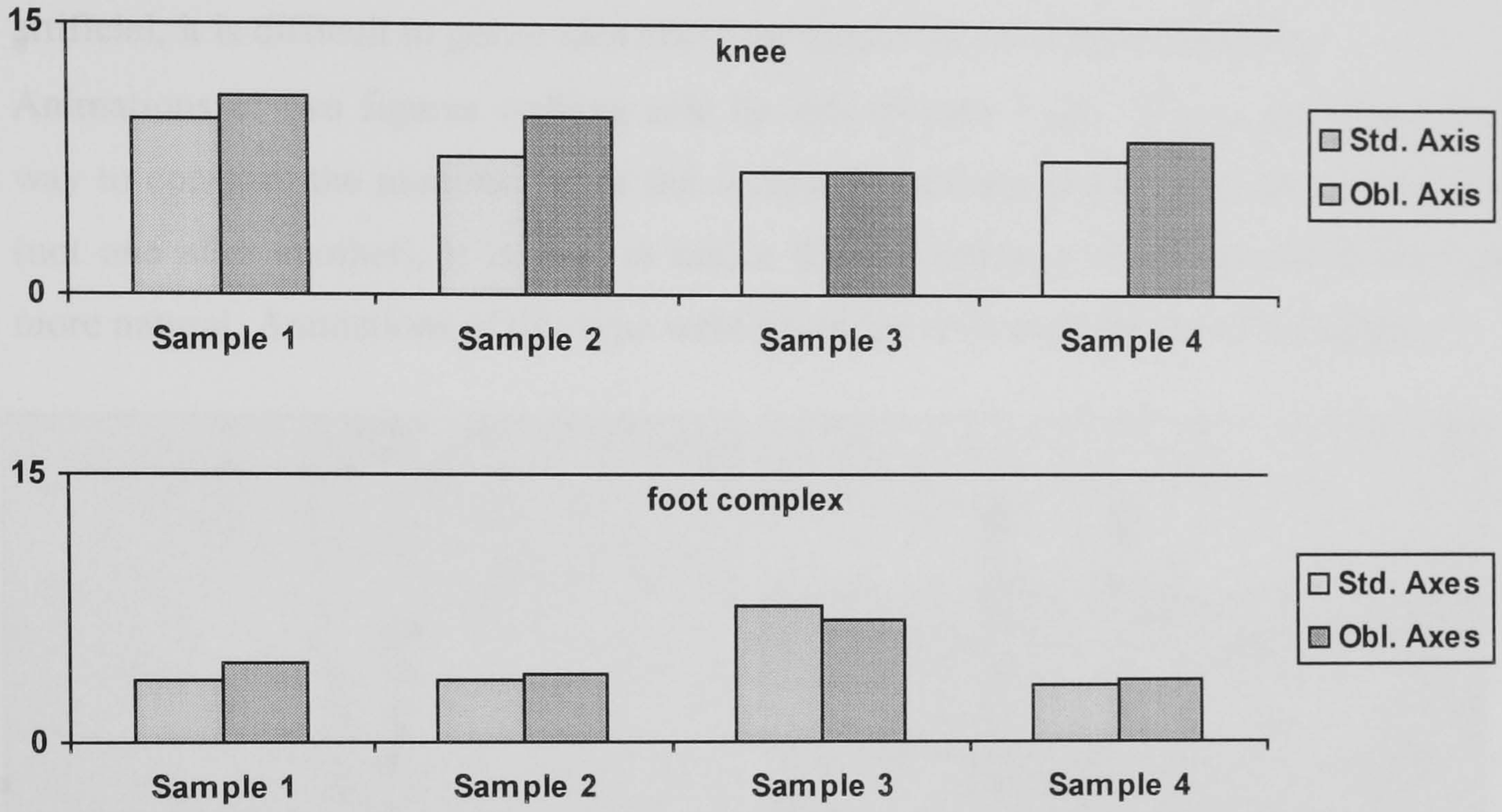


Figure 7.1: Approximation errors (in degrees) of the knee and the ankle

Similar results were obtained when analysing the correlation between the accuracy of the joint modelling and the accuracy of the resulting motion (amount of ground penetration, foot slipping, etc). Again, the choice of joint axes had some effect on the accuracy of the motion but the changes introduced were normally much smaller than the fluctuation of the accuracy between the motions, and even between the limbs.

The most important part of the evaluation is not a numerical but a visual analysis of the motions. For this, numerous animations were created and thoroughly examined both by the authors of the thesis and by external reviewers (Appendix C) in order to identify the effects of using different joint-motion models. All animations used for the analysis were produced by the proposed retargetting algorithm (Chapter 6) and were based on recorded walking data. These animations can be divided into three categories:

---

[1] The knee joint does not have a single flexion axis but has an instant axis of rotation. The axis used in the above experiments averaged this instant axis.

- Animations of a single figure. Several such animations (one for each set of joint parameters) are created for the same base motion and then viewed one after another. Such approach allows the detection of pronounced differences between the motions.
- Animations of two overlapping semi-transparent figures (Figure 7.2a). Using these animations one can compare the motions directly against each other, which allows one to notice subtle differences between them. However, since the animations look rather artificial, it is difficult to get an idea about the implications of the differences.
- Animations of two figures walking side by side (Figure 7.2b). This is the most natural way to compare the motions. Since the compared motions can be seen at the same time (not one after another), it is easy to notice their differences or select which one looks more natural. Animations of this type were given the reviewers for the examination.

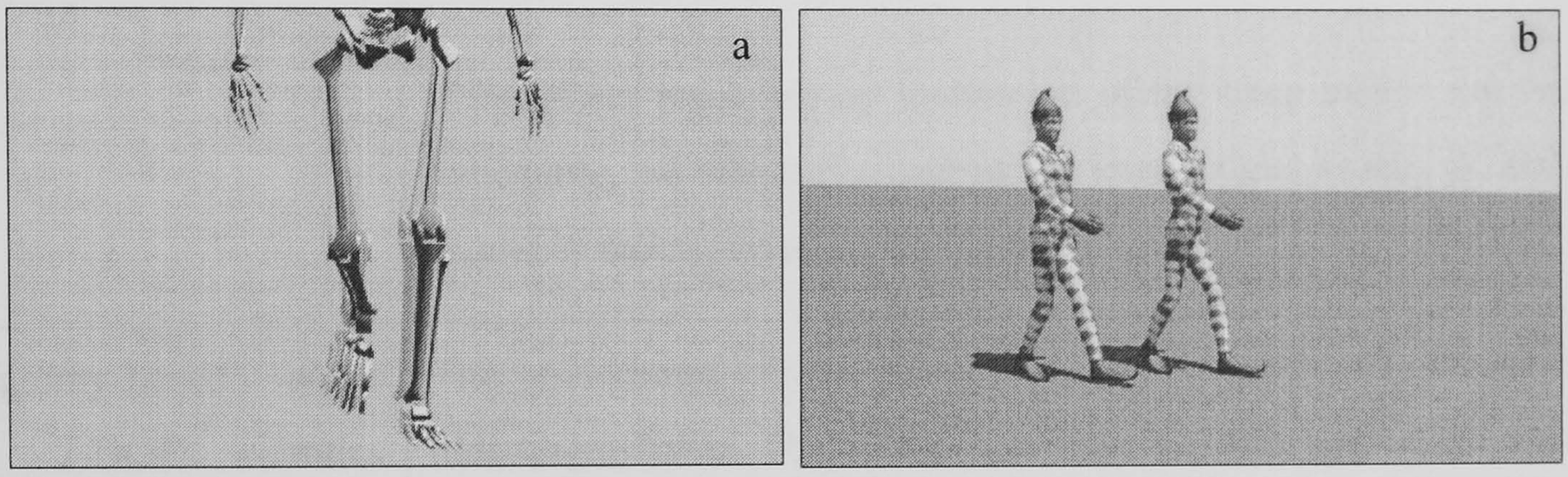


Figure 7.2: Examples of animations created to analyse the effect of biomechanics-related features of the model

The first conclusion made after examining the animations was that the influence of the joint model is less significant (visually) than many other aspects of modelling. In many cases, the motions produced were indistinguishable from each other when viewed separately; the effect of manipulating joint parameters could be seen only when the compared figures moved side by side or overlapped each other.

A more careful analysis of the test animations showed that some manipulations with joints parameters did produce a noticeable effect on the style of the motion. However, almost none of these could be called general because the effects produced varied from one motion to another. Observations made during this analysis are summarised in the table below.

Models	Analysis of differences
Anatomy-based knee vs. robotics-based knee	In some cases, the animations created using MC data show signs of discordance between different aspects of the motion. A typical example of such discordance is the inconsistency between the width of the walking base and the motion as a whole. This and some other aspects of the motion may be improved by changing inclination of the knee axis or by modifying the knee valgus angle. Use of an anatomy-based knee axis may also have a negative effect, producing an unpleasant artefact: interpenetration of the legs. However, this artefact may be eliminated if anatomical joints are used at the stage of pre-processing of raw MC data. If the effect produced by inclination of the knee axis is not very noticeable, the effect of changing the knee valgus usually is, and it produces the expected result (bow-leg walk).
Anatomy-based ankle vs. 2-DOF robotics ankle	The animations created using these two models are practically indistinguishable. The differences only become noticeable when the motions are compared in overlapping mode (Figure 7.2a).
Anatomy-based ankle vs. 1-DOF robotics ankle	Again, there are few visual differences between the animations. Though the uniaxial foot complex is an extremely rough approximation of the foot complex, the artefacts produced by such inaccuracies are not visually noticeable in the animation of walking.

Table 7.2: Comparison of motions produced using different figure models

It is necessary to add that these results can be only considered in the context of the animation technique that was used in this research, i.e. MC-based. The effects of combined motion may be more significant and obvious if the animations are created using a technique that is less dependent upon the measured data, such as a dynamics-based animation method. In MC-based methods, however, the motion is mainly defined by the input data and the effect of using biomechanics-based joint models is only achieved indirectly, via IK-based retargetting algorithm (Chapter 6) and the specialised motion-decomposition technique (Section 5.1.3). And, unfortunately, the kinematic redundancy of the skeleton model and the relatively low accuracy of the motion data does not allow full advantage to be taken of the use of the biomechanics-based joint model.

7.1.3 Coupling between Intervertebral Motions

Normally, the intervertebral joints would be considered independent of each other and, therefore, each such joint would add three unbound degrees of freedom to the figure model. The choice thus appears to be between a model that is inefficient (has too many DOF) or an inaccurate spine model consisting of two or three segments only.

Figure 7.3: Complex spine motion is distributed between the vertebrae automatically.

This problem was solved here by taking into account the interdependencies between the intervertebral joints: these interdependencies, which are defined by structural features of the column, allow bounding the intervertebral DOF (Figure 7.3), thus reducing the complexity of the model. Also, since the bounding expressions are derived from experimental data, the resulting motions are guaranteed to be anatomically correct.

A question that may arise is whether this model can be used to simulate any anatomically correct motion. The answer is no: the dependencies between the intervertebral joints are not absolutely rigid and can be bent, particularly if an external force is involved. However, many motions fit well into this model thus making it suitable for most applications.

The benefits of the above approach are easy to see: the model becomes simpler and easier to handle; the calculations are performed faster, and the animated motions look more natural. Of course, the described approach has some drawbacks. One drawback was encountered while "skinning" the figure with a polygonal mesh using a *Physique* modifier (Section 2.1.2.). Applying a mesh requires the animator to specify the correspondences between the mesh and each link. This procedure takes more time for a model with a 26-segment column than for a model with a two-segment spine.

7.2 Using Biomechanics Knowledge in MC Animation

Knowledge of human biomechanics can be used not only to construct a more accurate and efficient figure model but, most importantly, to create realistic and biomechanically accurate animations. The following sections summarise the main achievements of the research in this area.

7.2.1 Using Motion Analysis to Identify Constraints

Most retargetting algorithms require two types of input data: the initial estimation of the motion (i.e. the original MC data) and a set of constraints that define the goals of the retargetting. These constraints can be defined using two methods. The manual method requires the animator to specify all constraints by hand. Though this procedure is not as difficult as manual correction of the motion, it is still very time-consuming. The other method is to use an algorithm that can define the constraints procedurally. However, existing procedural methods do not go further than identifying the collision of the feet with the ground, and this is neither an accurate nor a reliable approach to the identification of constraints.

The proposed algorithm (Section 5.2) is much more accurate and reliable. Its idea is to apply Motion Analysis techniques to identify particular phases of the motion and use this information to derive the constraints. Using knowledge about the motion not only improves the accuracy and reliability of constraint identification, but it also makes the retargetting more flexible, as the algorithm can deal with the same constraints in different ways, depending upon the type of motion and the context in which a particular constraint takes place.

This method was successfully tested on walking motions. Using the knowledge about the gait, the algorithm can accurately decompose the motion into a sequence of strides, which are then subdivided into finer intervals, gait phases. This fine decomposition not only gives the required constraints, but also provides important information about their transitions, which is crucial for the creation of smooth realistic motions.

7.2.2 Knowledge-Based Retargetting

Existing retargetting methods [Gleicher, 98; Boulic, 90; Cho, 99] do not analyse the input motion capture data and, therefore, they deal with any motion in practically the same way. The only information that links the original motion and the retargetting algorithm is a set of uncorrelated constraints. This information is often insufficient for effective retargetting of complex and structured motions such as walking. As a result, the motions processed by such "blind" retargetting algorithms may have even more artefacts than the original motion had.

The proposed algorithm demonstrates that retargetting can benefit from knowing the type, the structure and the specific features of the processed motions (Chapter 6). Knowing the type of motion is important for choosing the optimal retargetting algorithm, as different motions may have different problems peculiar to them. Additionally, decomposition of a motion into strides allows the main retargetting algorithm to use different techniques for different parts of the

motion. In this way, the proposed correction algorithm uses different techniques for different stages of correction and for different parts of the motion: the global correction can be applied to any kind of motion, while the stride-level correction works with sub-intervals (*initial contact, flat foot*, etc) of individual walking and running strides. This idea can be extended to create techniques for a wider range of motions.

Motion Analysis allows an identification of significant features of the original motions that should be preserved during retargetting. This is also very important, because the mathematical methods used in retargetting algorithms often produce artificial-looking motions, and knowing features that are significant helps a compromise to be found between satisfying the constraints and producing biomechanically correct motions.

7.2.3 Perspectives of Motion Analysis in Character Animations

The use of Motion Analysis in Character Animation is not limited to retargetting. Among other areas that can benefit from MA are the creation of motion-data libraries, the automation of motion editing methods (blending, warping, montage), etc.

Considering how useful Motion Analysis can be, it seems strange that this area has been almost completely neglected by the CA researcher community. Motion Analysis certainly deserves more attention from researchers, and this project may provide some useful ideas for future studies in this area.

7.3 Future Developments

Though the proposed models and algorithms proved to be efficient and adequate for the posed problem, there are still many ways, in which they can be improved. Some ideas for improvements emerged during and after the development but were considered as secondary to the main task and were not taken into account at that time. Since the practical part of the project is intended as a general character-animation toolkit, which can be used in other projects, it would be useful to incorporate these improvements during any subsequent development.

One useful improvement to the figure model would be the support of quaternions as an internal representation of the joint motion. Though it was a well-thought-out decision to prefer the angle-axis format to the quaternion format for representing the joint motion (Section 4.5.1), practical experience has shown that using a single representation in all cases is not the best solution and combining both formats is likely to produce better results.

Another possible improvement is the creation of specialised joint models for the shoulder complex and the elbow-wrist complex. Since the project was oriented on modelling of the locomotion, these joints were considered secondary to the joints of the lower limb and no special models were designed for them. But, as experiments have demonstrated, none of the models created is adequate for simulating these joints and the use of specialised models is needed. Such models can be created in the framework of the existing architecture, which allows the modelling of the kinematic dependencies needed to define the complex character of motion in these joints.

The animation algorithm that was developed in the scope of the project needs some improvement too. One such improvement concerns the use of raw MC data (marker positions). The current implementation of the algorithm does not support such data and can use processed data only. Use of marker data, however, can widen possibilities of the proposed animation model. First of all, original marker positions are more accurate than the positions reconstructed from joint angles. Therefore, they can be used in applications that demand highly accurate data. For example, accurate foot marker data can be very useful for detecting foot-ground constraints. Second, the use of raw data means that the joint rotations are directly calculated from the marker positions using the target (biomechanics-based) model. This eliminates the intermediate joint model from the data-processing chain, which should reduce data-loss during the processing.

Another part of the retargetting algorithm that needs to be revised is the Inverse Kinematics engine. Though it successfully copes with the task of enforcing foot-ground constraints, it produces jerky motion that requires special post-processing (Section 5.4.2). Use of a more advanced IK algorithm that better exploits kinematic redundancies of the figure should reduce these artefacts. Hence the algorithm is likely to preserve important details of the original motion better.

Introducing a few improvements into the proposed models and algorithms is not the only task for future work in the area. An interesting research problem, which has not been approached here, is the integration of the biomechanics-based figure model with dynamics-based animation algorithms. Despite all the benefits of the proposed MC-based algorithm, it is limited by its dependency on the recorded motion. The anatomical and biomechanical properties of the actor reside in the recorded motion and they will still reside in it even if the motion is procedurally adjusted for an anatomically and biomechanically different model. The problem may be reduced if the data are captured and applied at a lower level i.e. using dynamics data and algorithms. In this case, the properties of the target, not the original model, will be crucial for the final animation. Unfortunately, this approach can only be realised when

dynamics-based animation methods become more general and are able to produce realistic motions.

7.4 Summary

The idea of the work presented in this thesis came from one biomechanical research, one of whose aims was to demonstrate that joint motion was not as simple as it seemed. The idea was to use biomechanical knowledge of human motion to create models for Character Animation and thus to learn how this knowledge can improve realism and universality of computer-generated animations of human locomotion. This orientation of the research was well reflected in the title of the thesis: "Animating human locomotion using biomechanics-based figure models". However, one cannot benefit from using an accurate improved model unless the methods that work with the model are adapted to it, particularly to its novel features. Therefore, creating an advanced knowledge-based algorithm for animating human locomotion goes side by side with creating a biomechanics-based figure model as the main objectives of this project.

The main body of the thesis began with an overview of Character Animation, which, in its turn, starts by introducing the cornerstone of character modelling, the skeleton model. The overview of the model suggested that the principles on which it is based are often too simplified, as a result of which, the figure model moves considerably differently from how a real human would move. It thus became clear that achieving the objectives of the project requires considerable improvements in this model.

An insight into human biomechanics, which allowed the creation an anatomically based figure model, was presented in Chapter 3. Topics from anatomy and biomechanics were covered, with particular attention being paid to the character of motion in the joints and the roles of particular joints in gait. Besides being used as a foundation for the figure model, this information was also crucial for creating the knowledge-based animation algorithm and was used throughout all phases of this algorithm.

From the beginning, this research was thought to be more practical than theoretical. Therefore, a system for testing all models and algorithms was needed. The requirements of such a system were formulated in the first section of Chapter 4. The rest of the chapter introduced the architecture of the model and describes its main elements.

When designing the system, close attention was paid to making it a convenient tool for research in Character Animation. Consequently, the system implemented represented more than just a realisation of the figure model and the corresponding animation mechanism. The

system was designed to be easily upgradable, so one could easily add, modify or replace different components (models, algorithms, etc) without worrying about affecting other components. The system is also equipped with various analysis tools and has several features aimed at simplifying development of animation algorithms.

There are several main approaches to animating computer characters. These include keyframe, kinematics-based, dynamics-based and MC-based animation techniques. The evaluation of these techniques, which were surveyed in Chapter 2, led to several conclusions. First, none of the above animation techniques can produce realistic results over a wide range of motions unless it is backed up by another technique. Second, the results produced by pure simulation methods are usually not satisfactory for most applications and the most realistic animations are generated with use of captured data. Bearing these conclusions in mind, Chapters 5 and 6 proposed an animation algorithm combining Motion Capture, Motion Analysis and Inverse Kinematics techniques to animate the figure model in a realistic and efficient way.

The proposed algorithm has several novel features that distinguish it from other MC-based algorithms. The cornerstone idea of these innovations is the same as it was with the figure model: study the subject and use the knowledge to improve the model/technique. In this case, it means that the animation algorithm is based on the knowledge of gait biomechanics.

The most significant innovation of the proposed animation method is the use of Motion Analysis techniques to analyse the captured data, classify it and extract information needed for the correction. Normally, retargetting (correction) algorithms consider all motion data in the same way and do not attempt to understand its type and structure. Some of these algorithms allow the user to supplement the captured data with some manually-defined data, such as constraints, keyframes; but, even in this case, the user-data cannot provide all the information needed to process the motion in the optimal way. The proposed algorithm, however, analyses the data and thus, not only can classify the motion, but it can also acquire its structure and get some other information needed for the correction algorithm.

The final chapter of the thesis summarised the novel features of the figure model and evaluated them in the context of animation. The first feature evaluated is support of different models of joint motion. Though it may seem to be just a matter of design, it is an important characteristic of the model and its significance is proved by the examples presented. The effects of modelling the combined motion on animation were discussed next. The corresponding section analysed the results of various practical experiments and gave some practical advice on using joints with this feature in animation. The last feature discussed was

modelling inter-joint kinematic dependencies. This evaluation was continued by a brief review of the proposed animation algorithm. A description of possible future work and this summary concluded the thesis.

To provide a better picture of the results of this research, the thesis is supplemented by a CD, which contains various animations and still images produced using the algorithms described. Additional videos and images, along with all source code created in scope of this project, are available on the website of the project: http://www.mk.dmu.ac.uk/~asavenko/project/

References

3D Studio Max 3.1. Software Development Kit. (1999) [CD-ROM]

Aggarwal J., Cai, Q. (1999). Human Motion Analysis: a Review. Computer Vision and Image Understanding, 73(3), 428-440. Available on the World Wide Web at http://citeseer.nj.nec.com/aggarwal99human.html

Amaya, K., Bruderlin, A., Calvert, T. (1996). Emotion from Motion, Proceedings of Graphics Interface '96, 222-229. Available on the World Wide Web at http://www.dgp.toronto.edu/gi/gi96/proceedings/papers/ABC/Amaya.ps.gz

Aubel, A., Thalmann, D., (2000). Realistic Deformation of Human Body Shapes, Proceedings of Computer Animation and Simulation 2000, 125-135.

Ayyappa, E. (1997). Normal Human Locomotion, Part 1: Basic Concepts and Terminology. Journal of Prosthetics and Orthotics. 9(1), 10-17.

Badler, N., Manoochehri, K., Walters, G. (1987). Articulated Figure Positioning by Multiple Constraints, IEEE Computer Graphics and Animation, 7(6), 28-38.

Boulic, R., Thalmann, D. (1992). Combined Direct and Inverse Kinematic Control for Articulated Figures Motion Editing, Computer Graphics Forum, 11(4), 189-202. Available on the World Wide Web at http://ligwww.epfl.ch/~thalmann/papers.dir/CGF.comb.dir.inv.pdf

Boulic, R., Fua, P., Herda, L., Silaghi, M., Monzani, J., Nedel, L., Thalmann, D. (1998). An Anatomic Human Body for Motion Capture, Proceedings of EMMSEC 98. Available on the World Wide Web at http://ligwww.epfl.ch/~thalmann/papers.dir/EMMSEC98.pdf

Boulic, R., Magnenat-Thalmann N., Thalmann, D. (1990), Coach-Trainee: A New Methodology for the Correction of Predefined Motions, Proceedings of Eurographics Workshop on Animation and Simulation, E1-E14. Available on the World Wide Web at http://ligwww.epfl.ch/~thalmann/papers.dir/EGWK91.coach.pdf

Boulic, R., Magnenat Thalmann N., Thalmann, D. (1990). A Global Human Walking Model with Real-Time Kinematic Personification. The Visual Computer, 6, 344-358.

Bruderlin, A., Calvert, T. (1989). Goal-Directed, Dynamic Animation of Human Walking, Computer Graphics, 23(3), 233-242.

Bruderlin, A., Calvert, T. (1993). Interactive Animation of Personalized Human Locomotion, Proceedings of Graphics Interface '93, 17-23.

Bruderlin, A., Williams, L. (1995). Motion Signal Processing, Proceedings of SIGGRAPH '95, 97-104.

Character Studio™ R2. User's Guide. (1998)

Choi, K., Park, S., Ko, H. (1999). Processing Motion Capture Data to Achieve Positional Accuracy. The Journal of Graphical Models and Image Processing, 260-273. Available on the World Wide Web at http://graphics.snu.ac.kr/research/pmcd/

Computer Graphics World (Editors). (2002). Star Wars: Episode II: The Impact of the Effects. Computer Graphics World (internet magazine). Available on the World Wide Web at http://cgw.pennnet.com/Articles/Article_Display.cfm?Section=OnlineArticles&SubSection=Display&PUBLICATION_ID=18&ARTICLE_ID=148096

Davis, J., Bobick, A. (1998). A Robust Human-Silhouette Extraction Technique for Interactive Virtual Environments. Proceedings of Modelling and Motion capture Techniques for Virtual Environments, 12-25.

Delaney, B. (1998). The Mystery of Motion Capture, IEEE Computer Graphics and Applications, 18(5), 14-19.

Faure, F., Debunne, G., Cani-Gascuel M., Multon, F. (1997). Dynamic Analysis of Human Walking. Proceedings of the Eurographics Workshop on Animation and Simulation, 53-65. Available on the World Wide Web at http://www-imagis.imag.fr/~Francois.Faure/papers/humanWalking.html

Finkelstein, A., Salesin, D. (1994). Multiresolution Curves. Proceedings of SIGGRAPH '94, 261-268.

Fua, P., Plänkers, R., Thalmann, D. (1998). Realistic Human Body Modeling, Proceedings of 5th International Symposium on the 3-D Analysis of Human Movement. Available on the World Wide Web at http://ligwww.epfl.ch/~fua/papers/fua-et-al-ishm98.pdf

Girard, M., Maciejewski, A. (1985). Computational modeling for the computer animation of legged figures, Proceedings of SIGGRAPH '85, 263-270.

Gleicher, M. (1998). Retargetting motion to new characters. Proceedings of SIGGRAPH '98, 33-42.

Hadap, S., Magnenat-Thalmann, N., (2001). Modeling Dynamic Hair as a Continuum. Computer Graphics Forum, 20 (3), 329-338. Available on the World Wide Web at http://www.miralab.unige.ch/newMIRA/ARTICLES/MDHC.pdf

Hodgins J., Wooten W., Brogan D., O'Brien J. (1995). Animating Human Athletics. Proceedings of SIGGRAPH '95, 71-78. Available on the World Wide Web at http://www.cs.gatech.edu/gvu/animation/papers/sig95.pdf

Hodgins, J. (1996). Three-Dimensional Human Running, Proceedings of the IEEE Conference on Robotics and Automation, 3271-3276. Available on the World Wide Web at http://www.cc.gatech.edu/gvu/animation/papers/icra96.pdf

Hreljac A., Stergiou N. (2000). Phase determination during normal running using kinematic data, Medical and Biological Engineering and Computing, 38(5), 503-506.

Kapandji, I. A. (1985). Physiologie Articulaire, Tome 2. ISBN 2-224-01052-4.

Ko, H., Badler, N. (1993). Straight line walking animation based on kinematic generalization that preserves the original characteristics, Proceedings of Graphics Interface '93, 9-16.

Ko, H., Badler, N. (1996). Animating Human Locomotion with Inverse Dynamics. IEEE Computer Graphics and Applications, 16(2), 50-59. Available on the World Wide Web at http://graphics.snu.ac.kr/research/publication.html

Laszlo, J., Van de Panne, M., Fiume, E. (1996). Limit Cycle Control and its Application to the Animation of Balancing and Walking, Proceedings of SIGGRAPH '96, 155-162. Available on the World Wide Web at http://www.dgp.toronto.edu/people/jflaszlo/sig96.ps.gz

Luttgens, K., Wells, K. (1982). Kineosiology. Scientific basis of Human Motion. 7th Edition, Saunders College Publishing.

Magnenat-Thalmann, N., Thalmann, D. (2001). Deformable Avatars. ISBN 0-792-37446-0.

Magnenat-Thalmann, N., Thalmann, D. (1996). A System for the Animation of Virtual Humans. Open Systems (internet magazine). Available on the World Wide Web at http://ligwww.epfl.ch/~thalmann/papers.dir/Journal.Russe.pdf

Marsden, J. E., Tromba A. (1996). Vector Calculus. ISBN 0-716-72432-4

Boulic, R., Mas, R., Thalmann, D. (1997). Complex Character Positioning Based on a Compatible Flow Model of Multiple Supports, IEEE Transactions in Visualization and Computer Graphics, 3(3), 245-261.

Maurel, W. (1998). 3D Modeling of the Human Upper Limb including the Biomechanics of Joints, Muscles and Soft Tissues, PhD thesis, Laboratoire d'Infographie, Ecole Polytechnique Federale de Lausanne. Available on the World Wide Web at http://ligwww.epfl.ch/people/maurel/public_html/

Multon, F., France, L., Cani-Gascuel, M., Debunne, G. (1999). Computer Animation of Human Walking: a Survey, Journal of Visualization and Computer Animation, 10, 39-54. Available on the World Wide Web at http://www.inrialpes.fr/bip/people/france/stuff/survey.ps.gz

Nebel, J. C. (1999). Keyframe Interpolation with Self-Collision Avoidance. Proceedings of Eurographics Workshop on Computer Animation and Simulation, 77-86.

Nedel, L., Thalmann, D. (1998). Modeling and Deformation of Human Body using an Anatomy-Based Approach, Computer Animation '98, 34-40. Available on the World Wide Web at http://ligwww.epfl.ch/~thalmann/papers.dir/CA98.muscles.pdf

Norkin, C., Levangie, P. (1983). Joint Structure and Function. A Comprehensive Analysis. ISBN 0-8036-6576-8.

Poser 4.0 Review. (1999). MacUser (internet magazine). Available on the World Wide Web at http://www.macuser.co.uk

Rose, C., Guenter, B., Bodenheimer, B., Cohen, M. (1996). Efficient Generation of Motion Transitions Using Spacetime Constraints. Proceedings of SIGGRAPH '96, 147-154. Available on the World Wide Web at ftp://ftp.research.microsoft.com/users/hfap/final.ps.gz

Reinschmidt, C., van den Bogert, A., Nigg, B., Lundberg, A., Murphy, N. (1997). Effect of Skin Movement on the Analysis of Skeletal Knee Joint Motion During Running. Journal of Biomechanics, 30(7), 729-732.

Scheepers, F., Parent, R., Carlson, W., May, S. (1997). Anatomy-Based Modeling of the Human Musculature. Proceedings of SIGGRAPH'97, 163-172.

Steketee, S. N., Badler, N. (1985). Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control, Proceedings of SIGGRAPH '85, 255-262.

Sun, W., Clapworthy, G. (1997). A Wavelet-Based Description of Bipedal Locomotion for use in Virtual Worlds, Proceedings of 5th International Conference on CAD/Graphics, 1, 145-151.

Unuma, M., Anjyo, K., Takeuchi, R. (1995). Fourier Principles for Emotion-based Human Figure Animation, Proceedings of SIGGRAPH '95, 91-96.

Van Der Bauwhede, J., (1996) Wheeless' Textbook of Orthopaedics (online). Available on the World Wide Web at http://www.medmedia.com/med.htm

Van Sint Jan, S., Clapworthy, G., Rooze, M. (1998). Visualization of Combined Motions in Human Joints. IEEE Computer Graphics & Applications, 18(6), 10-14.

Van Sint Jan, S., Salvia, P., Clapworthy, G., Rooze, M. (1999). Joint-Motion Visualization Using Both Medical Imaging and 3D-Electrogoniometry. Proceedings of 17th Congress of the International Society of Biomechanics. Available on the World Wide Web at http://isb.ri.ccf.org/tgcs/iscsb7/abstracts/vansintjan.pdf

Vasilonikolidakis, N., Clapworthy, G. (1991). Design of Realistic Gaits for the Purpose of Computer Animation. Computer Animation, 101-114, ISBN 0-387-70077-3.

Welman, C. (1993). Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation. MSc thesis, School of Computer Science, Simon Fraser University. Available on the World Wide Web at ftp://fas.sfu.ca/pub/cs/theses/1993/ChrisWelmanMSc.ps.gz

White, A. (1990). Clinical biomechanics of the spine. ISBN 0-397- 50720-8.

Whittle, M. (1991). Gait Analysis: An Introduction. ISBN 0-750-60045-4.

Wilhelms, J., Van Gelder, A. (1997). Anatomically Based Modeling. Proceedings of SIGGRAPH '97, 173-180. Available on the World Wide Web at http://citeseer.nj.nec.com/

Witkin, A., Kass, M. (1988). Spacetime Constraints, Computer Graphics, 22(4), 159-168.

Witkin, A., Popovic, Z. (1995). Motion Warping. Proceeding of SIGGRAPH '95, 105-108. Available on the World Wide Web at http://www.ri.cmu.edu/pub_files/pub1/witkin_andrew_1988_1/witkin_andrew_1988_1.pdf

Zeltzer, D. (1982). Motor Control Techniques for Figure Animation. IEEE Computer Graphics and Application, 53-59.

Zhao, J., Badler, N. (1994). Inverse kinematics positioning using nonlinear programming for highly articulated figures, ACM Transactions on Graphics, 13(4), 313-336. Available on the World Wide Web at http://www.acm.org/pubs/toc/Abstracts/tog/195827.html

Appendix A: Inverse Kinematics

If the task of Forward Kinematics is to calculate the positions and the orientations of links given the joint angles, the task of Inverse Kinematics is the opposite. One specifies the position and the orientation of the link (end-effector) and the task is to find the values of the joint angles that realise the specified position of the link.

In order to find a solution for the IK task, researchers sought assistance from Robotics, which provides several methods to solve the Inverse Kinematics problem. IK was first proposed for use in Character Animation by Girard and Maciejewski [1985]. They used the Inverse Jacobian method, which later became the most widely used IK method in Computer Animation. This algorithm works as follows:

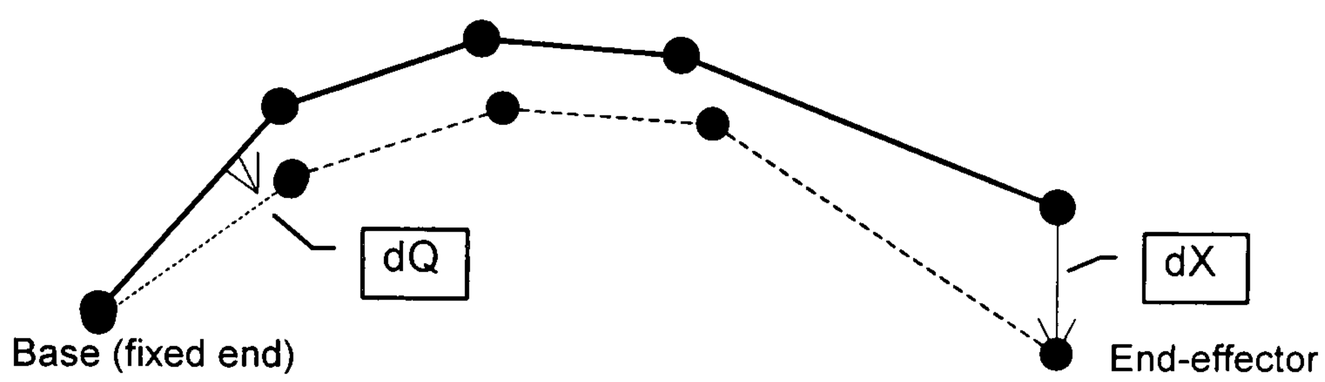


Figure A1: An initial and final state of a kinematics chain

Given the initial state of the chain Q and the differential of the end-effector position dX, the Inverse Jacobian algorithm calculates the differential of the chain state dQ that realises the new end-effector position $X + dX$. This method does these calculations in several steps:

1. First, an equation that relates the velocity of the end-effector and the joint velocities is calculated and linearized: $x' = J(q) \cdot q'$
2. The next step is to invert matrix J, which is called the *Jacobian*. There is no difficulty if the matrix is square and non-singular. However, in most cases the chain is redundant (i.e. the same position of end-effector can corresponds to several states of the chain) and the matrix is singular. In this case a numerical algorithm is used and a criterion is introduced to select the most appropriate solution among the set of possible solutions.
3. Inverted matrix J* is used to find the derivative of the chain state: $q' = J^* \cdot x'$. The new chain state Q_i can be found by adding the derivative to the original state ($Q_i = Q_{i-1} + q'$) or using more accurate integration methods.
4. These steps are repeated until an acceptable solution is found (i.e. $| X(Q) - X_{target} | <$ Threshold).

The cause of most of the problems of IK-based algorithms is redundancy of kinematics chains i.e. there can be an infinite number of solutions that realise the desired position of the end-

effector. The most common way to address this is to use an optimisation technique to find the optimum solution [Badler et al, 1987; Zhao et al, 1994]. The optimisation metrics differ from one method to another. They can include energy metrics, penalty metrics that incorporate constraints, etc. Since there are no scientific basis for these metrics (and it is unlikely there will be any in the closest future), all of these empirical metrics require a lot of experimenting and manual tuning to get realistic results. In addition, it is not guaranteed that metrics tuned for one motion will work for another one.

The second drawback of IK-based methods emerges when the change of the end-effector position is large (for instance, while looking for the limb trajectory between two steps). Though the algorithm guarantees that the final position of the end-effector is valid it does not guarantee the continuity of the solution along the trajectory.

These disadvantages do not allow the use of IK as the base method for the simulation of the locomotion. However, IK is still the best method when it is required to perform small corrections of the motion and so it was used in the project to assert the validity of the foot-ground constraints.

Implementation of Inverse Jacobian Algorithm

The mathematical apparatus of Inverse Jacobian algorithm is complex but it can be divided into several modules (classes). Our implementation of the algorithm is based on the following classes:

- Class Jacobian implements the functionality of Jacobian matrices. It has methods for creating the matrix, filling its cells and performing arithmetic operations (multiplication) with the matrices.

- Class JacobianI corresponds to Inversed Jacobian. It is responsible for the inversion of the Jacobian objects and is used to calculate the differentials of kinematic-chain states.

- Auxiliary classes JacobianBase, JacobianT and MatrixNxN are private classes used in calculations.

Here are the prototypes of these classes:

```
struct JacobianBase {
  float (*comp)[6];     // the structure corresponds to the transposed matrix
  int    N;             // number of DOF
  int    EDOF;          // number of End-Effector DOF (3 or 6)
  int    mask;          // end-effector properties (enum EEStates)
  };
```

```
class CoreExport Jacobian : protected JacobianBase {
  friend JacobianI;

  // inverse jacobian
  JacobianI *iJacobian;
  bool       inverseValid;

  int currentDOF; // number of added DOF

  public:
    // constructors
    Jacobian(int n, int initMask = EE_ALL);
    Jacobian(const Jacobian& j);
    ~Jacobian();

// conversion
operator const JacobianT&();
JacobianI& Pseudoinverse(double dumpingFactor, bool recalculate);

void operator =(const Jacobian& j) { assert(false); }

// access
//inline double at(int row, int col);
int  size() const;
int  GetEndEffectorMask() const;
void SetEndEffectorMask(int m);

// creating Jacobian
void Reset() { inverseValid = false; currentDOF = 0; }
void AddDerivatives(const Point3& pos, const Point3& euler);
void AddDerivatives(const Point3& deriv);

// arithmetics
    //void product(const float *q, EEState state, Point3& x) const;
    bool Ready() const; // true if all derivatives are defined

    void ApplyWeights(const float *weights);
  }; // class Jacobian


class CoreExport JacobianT : protected JacobianBase {
  friend JacobianI;

  private:
    JacobianT()  { assert(false); }
    ~JacobianT() { assert(false); }

    void operator =(const JacobianT& j) { assert(false); }

  public:
    int size() const { return N; }

    // arithmetics
    void product(const Point3& pos, const Point3& orient, float q[]) const;
    void product(const Point3& diff, float q[]) const;
    void sqr(MatrixNxN& m) const; // Jt * J

  private:
    operator const float *() const { return (float *)comp; }
  }; // class JacobianT


class CoreExport JacobianI {
  private:
    Jacobian& jacobian;
    double (*comp)[6];    // components of the matrix
```

```
  int N;                    // number of DOF in the chain

  bool overflowFlag;        // result of the inversion

  MatrixNxN *matrix;        // auxiliary data

  // constructors
  JacobianI(Jacobian& j);
  ~JacobianI();

  void operator =(const JacobianI& j) { assert(false); }

public:
  // arithmetics
  bool product(const Point3& v, const Point3& orien, float q[]) const;
  bool product(const Point3& diff, float q[]) const;
  bool calculate(double dumpingFactor);
  bool Projection(MatrixNxN& m) const; // Pr(J) = I - J+ * J
  bool IsOverflow();
}; // class JacobianI
```

An Inverse Kinematics algorithm cannot the problem of locating the end-effector by itself – the algorithm has to be integrated into a high-level control scheme. This scheme is responsible for controlling the convergence of the algorithm, calculating the accuracy, choosing the step size, detecting singularities, etc. Therefore, a careful implementation of this scheme is crucial for any IK-based method. The actual code of the scheme used in our project can be found in the file *correctionIK.cpp* of the Figure project [Figure Plug-In, 2001]. This scheme is specially designed for manipulating the lower limb and takes into account different types of foot constraints and singularities, which are typical for these kind of kinematic chains.

Below is simplified pseudocode of an IK algorithm, which gives an idea of how to implement an IK-controlling scheme with the introduced IK classes.

```
Jacobian jacobian(numberOfDOF);

do {
  // fill the jacobian
  for ( all joints in the chain)
   // calculate derivatives for every joint (dof) and add them to jacobian
   joint.ComputeDerivatives(dofValues, jacobian, eePos, matrix);
   // specify optional weights for the derivatives
   jacobian.ApplyWeights(dofWeights);

  // invert the jacobian and calculate the derivative of the chain state
  JacobianI& ijacobian = jacobian.Pseudoinverse(0.0%5, true);
  ijacobian.product(posOffset, orientOffset, derivative);

// integration
for ( all dofs in the chain)
  dofValues[i] += derivative[i];

// recalculate the accuracy (forward kinematics)
accuracy = RecalculateAccuracy();

} while (accuracy > threshold);
```

Appendix B: Joint Data

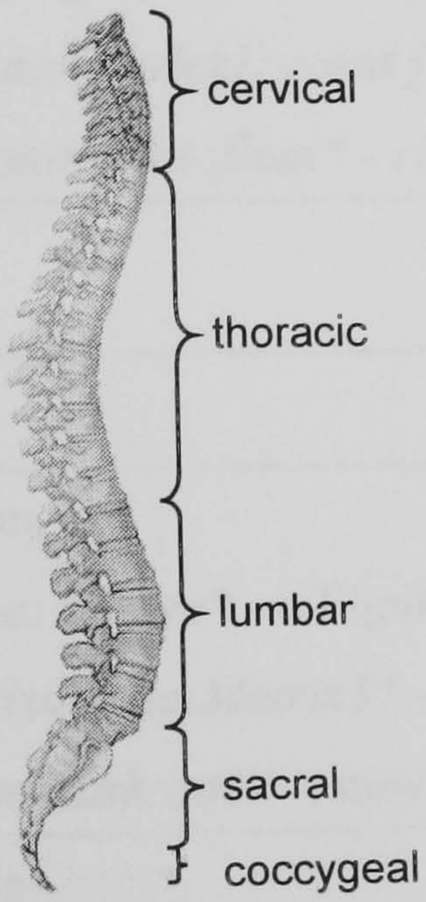
The values presented here (normal ranges of motion, inclination of the axes) are used as default parameters for the joints of the figure model. All of these values are averages calculated from numerous measurements and were taken from the biomechanics literature [Norkin & Levangie, 1983]. Note also that any of these values can vary significantly among individuals and therefore these averages should be used only as guides.

	Articulation	Normal Range	Axis
Hip	Flexion / Extension	$100^0/35^0$	Coronal
	Abduction / Adduction	$45^0/25^0$	Anterior-posterior
	Int. / Ext. Rotation	$25^0/35^0$	Vertical
Knee	Flexion / Hyperextension	140^0 / 1^0	
	Abduction / Adduction (passive motion only)	Varies greatly among persons	Anterior-posterior
	Int./Ext. Rotation	$30^0/40^0$ (at 90^0 of flexion)	Vertical
Foot	Plantar / Dorsiflexion	$20^0/50^0$	Mainly Coronal (25^0 with frontal plane and 10^0 with transversal)
	Inversion / Eversion	$10^0/20^0$	45^0 with frontal plane and 45^0 with transversal
Toe	Flexion / Extension	$30^0/90^0$	
Spine	Flexion / Extension	$90^0/90^0$	Coronal
	Lateral Flexion	$90^0/90^0$	Anterior-posterior
	Rotation	$45^0/45^0$	Vertical
Shoulder	Flexion / Extension		Coronal
	Abduction / Adduction		Anterior-posterior
	Int. / Ext. Rotation		Vertical
Elbow	Flexion	$150^0/0^0$	85^0 to Vertical
Elbow-Wrist	Pronation/ supination	There was no intention of creating anatomical models for the joints of the upper body and limbs. So the motion in the lower arm were modelled using the simplest, ball&socket, joint.	
Wrist	Flexion/extension		
	Deviation		

Table A1: Standard joint parameters

Note about the range of motion: it is assumed that the limits of the motion are never reached in normal locomotion. This means that, for most of the articulations, the information about their range of motion is not really used in simulation (motion correction). The exceptions are knee flexion, ankle dorsiflexion and toe flexion. For all other motions, the ranges of motion are used for control and analysis purposes only.

The following table represents the coupling ratios between the vertebrae in the spinal column (White, 1990). These ratios are used in the spine controller to distribute the motion between the vertebrae.



	Region		Flexion	Lateral Flexion	Rotation
	Cervical	C1	18%	7%	55%
		C2	9%	12%	5%
		C3	13%	14%	10%
		C4	18%	20%	10%
		C5	18%	20%	10%
		C6	15%	17%	8%
		C7	9%	8%	2%
	Thoracic	T1	2.5%	5.5%	10%
		T2	2.5%	5.5%	10%
		T3	2.5%	5.5%	10%
		T4	2.5%	5.5%	10%
		T5	4%	5.5%	10%
		T6	4%	5.5%	8.5%
		T7	4%	5.5%	8.5%
		T8	4%	5.5%	7.5%
		T9	4%	5.5%	5%
		T10	6%	6.5%	2.5%
		T11	7.5%	8.5%	2.5%
		T12	7.5%	7.5%	2.5%
	Lumbar	L1	7.5%	5.5%	2.5%
		L2	10%	5.5%	2.5%
		L3	10%	7.5%	2.5%
		L4	10%	5.5%	2.5%
		L5	11%	2.5%	1.25%
	Sacral		0.5%	1%	1.25%

Table A2: Ratios of motion coupling between vertebrae

Appendix C: Class Pose

Below is a simplified diagram of class Pose, which is used to automate the task of performing Forward Kinematics manipulation with the figure model.

Class PoseData

Attributes

values: float array

Methods

PoseData() - constructors

Read(buffer) / Write(buffer) – IO operations

operator [index]: const float & – access operator

operator float :float - conversion operator*

Class Pose

Attributes

*figure: Figure** - a Figure object associated with this Pose object

*matrixCache:Matrix3** - an array where the resulting matrices are cached

cacheMask:int64 – state of the cache elements

Methods

Pose() – constructors

LoadValues()– load DOF data to the internal storage

CopyToControllers(time) – copy data from the internal storage to Figure controllers

Reset() - reset DOF data

SetValue(dof, value)/AddValue(dof, value) - set/add value of a DOF

GetMatrix(joint, matrix, method):Matrix3& - get transformation of a joint

GetPosition(joint, point, method): Point3& - get position of a joint

GetOrientation(joint, point, method): Point3& - get orientation of a joint (euler angles)

GetHeelPosition(side, point, method): Point3& - get heel position

GetToesPosition(side, point, method):Point3& - get toes position

GetState(side, ignoreGround):int - get foot state (heel contact, flat-foot , etc)

ApplyContraintsToFeet(constraints) - apply constraints (heel contact, flat-foot , etc)

Appendix D: Questionnaire

Below is the questionnaire that was used for evaluating the correction algorithm (Chapter 6) and for analysing the effects of produced by different aspects of the figure model on animation (Chapter 7).

Questionnaire

Name: _____

The aim of this questionnaire is to evaluate a new approach to creating animations of human locomotion. Comparison of motions generated using different algorithms and kinematic models will help us to understand how these affect the visual impression of animation.

Please look at the animations and evaluate how realistic are the motions. For each motion you should put two pairs of marks. The first mark is needed to evaluate the initial general impression from the animations. Please put this mark after watching the animation one or two times. The second mark should be put after you study the motions more scrupulously. Use the comment section to describe the visual difference between the motions and to mention any visual artefacts.

Animation #1

	Green	violet
General mark		
Detailed mark		
Comments		

Animation #2

	green	violet
General mark		
Detailed mark		
Comments		

In total there were 8 animations: 4 for evaluation of the retargetting method and 4 for analysis of the figure model. Two example images from these animations are shown on Figure D2.

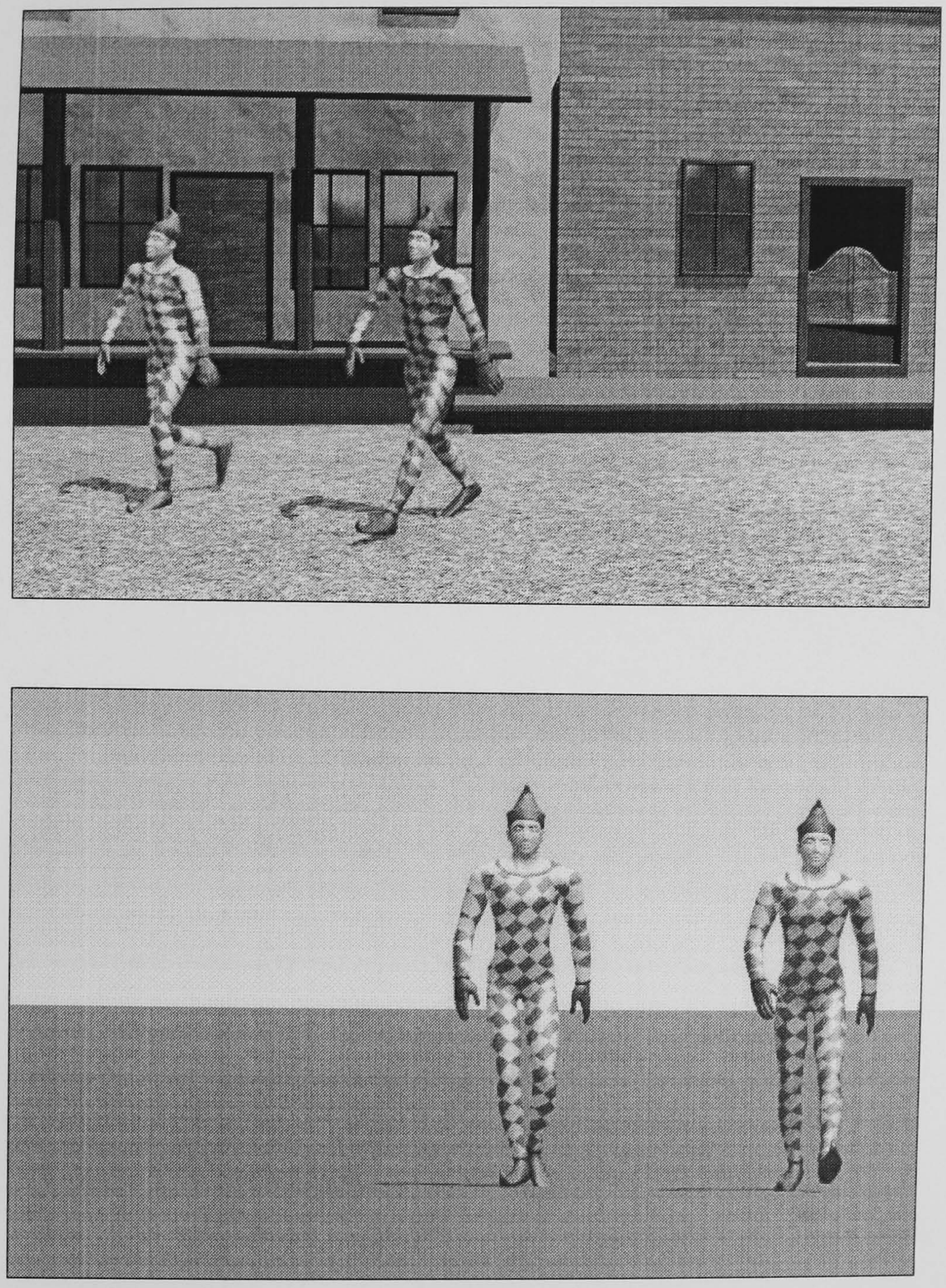


Figure D2: Examples of the testing animations