REAL-TIME PARAMETERIZED LOCOMOTION GENERATION

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BİLKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Muzaffer Akbay

September, 2008

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Uğur Güdükbay (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Enis Çetin

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Özgür Ulusoy

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

REAL-TIME PARAMETERIZED LOCOMOTION GENERATION

Muzaffer Akbay
M.S. in Computer Engineering
Supervisor: Assoc. Prof. Dr. Uğur Güdükbay
September, 2008

Reuse and blending of captured motions for creating realistic motions of human body is considered as one of the challenging problems in animation and computer graphics. Locomotion (walking, running and jogging) is one of the most common types of daily human motion. Based on blending of multiple motions, we propose a two-stage approach for generating locomotion according to user-specified parameters, such as linear and angular velocities. Starting from a large dataset of various motions, we construct a motion graph of similar short motion segments. This process includes the selection of motions according to a set of predefined criteria, the correction of errors on foot positioning, pre-adjustments, motion synchronization, and transition partitioning. In the second stage, we generate an animation according to the specified parameters by following a path on the graph during run-time, which can be performed in real-time. Two different blending techniques are used at this step depending on the number of the input motions: blending based on scattered data interpolation and blending based on linear interpolation. Our approach provides an expandable and efficient motion generation system, which can be used for real time applications.

Keywords: Animation, data scattered interpolation, blending, locomotion.

ÖZET

GERÇEK ZAMANLI PARAMETRİK GEZME HAREKETİ TÜRETİLMESİ

Muzaffer Akbay
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Uğur Güdükbay
Eylül, 2008

Gerçekçi insan vücut animasyonu yapımında özel donanım yardımıyla yakalanmış hareketlerin tekrar kullanımı ve karıştırılması animasyon ve bilgisayar grafiğinin en zor problemlerinden biridir. Yürüme, hızlı ve yavaş koşma günlük insan eylemleri arasında en sık kullanılanlardandır. Bu tezde, bu tür hareketlerin kullanıcı tarafından tanımlanmış açısal ve doğrusal hız gibi parametreler doğrultusunda üretilmesi için çoklu karıştırmaya dayalı iki aşamalı bir yöntem önerilmiştir. Birinci aşamada, geniş bir veri tabanından başlayarak benzer ve kısa hareketlerden oluşan bir çizge oluşturulmaktadır. Bu aşama, bahsi geçen veri tabanından hareket ayıtlanması, ön ayarlamalar, hata düzeltilmesi, hareketlerin senkronize edilmesi, ve hareket geçişlerine göre kısımlara ayrılmasından oluşur. İkinci aşamada, oluşturulan çizge üzerinde bir yol takip edilerek verilen parametrelere göre animasyon gerçek zamanda üretilir. Bu aşamada, karıştırmada kullanılacak hareket sayısına göre iki ayrı yaklaşım kullanılır: *saçılmış veri aradeğerlemesine dayalı karıştırma* ve *doğrusal aradeğerlemeye dayalı karıştırma*. Tanımlanan sistem genişletilebilir ve etkili bir sistemdir, ve gerçek zamanlı uygulamalarda kullanılabilir.

Anahtar sözcükler: Canlandırma, saçılmış veri aradeğerlemesi, karıştırma, gezme hareketi.

Acknowledgement

I would like to acknowledge the supervision of Assoc. Prof. Dr. Uğur Güdükbay who supported and guided my research on this topic.

I would like to express my gratitudes to Prof. Dr. Enis Çetin and Prof. Dr. Özgür Ulusoy for kindly accepting to spend their valuable time to evaluate my thesis.

I am grateful to my family for supporting my academic and social education, and I would like to thank H. Emre Kale and E. Büşra Çelikkaya for their comments and support on this study.

The data used in this project was obtained from `http://mocap.cs.cmu.edu`. The database was created with funding from NSF EIA-0196217.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Early animation techniques consist of displaying images consecutively in order to achieve motion. In the last twenty years, with the rapid development in computer technology, the studies on animation technologies, aiming for more realistic motions with more control and flexibility for the animators, has become popular. Articulated figure animation is used extensively in making of many movies and games. Motion capture technology is one of the most common methods for obtaining articulated figure animation. The use of motion capture in animation has begun in the late 1970s, and nowadays it spreads quickly.

Motion Capture (MoCap) technology is a recording technology, and it is used to record the behaviors of actors. After the recording, these recordings are converted into a virtual 3D environment for further editing. In general, MoCap targets the motions of the actors, not their physical appearance. However, in recent studies for muscular simulation, the skin deformations of the actors are also captured [2]. In movie industry, this technique is used for creating scenes that are physically impossible and also for projecting the motions of the stunts onto main actors of the scenes. In game industry, MoCap is mainly used for acquiring realistic motions, such as martial arts and athletics, and for applying them on 3D models.

There have been various techniques employed for motion capturing, which

made its first sight as a photogrammetric analysis tool in biomechanics research in 1970s and 1980s.

Optical systems: Optical systems use several image sensors (cameras) for projecting the captured data from all camera angles onto 3D environment. These systems traditionally perceive the motions by tracking the special markers attached to various places on the actors' body. The raw data produced by optical systems generally includes positions of the markers in 3D space. Then, this data is processed and converted into a hierarchical representation with the joint angles and the root positions. For instance, the markers on hip, femur and tibia are used in acquiring angle of the knee. There are various types of markers. *Passive markers* reflect back the light that is generated near camera lens. For calibration, a bright object is placed at a known position and positioning of the other markers are measured with respect to this object. *Active markers* emit their own light, rather than reflecting an external light. The LEDs on the markers are blinked very quickly one after another. By calibrating the capturing frequency with blinking frequency, the markers can be identified. In place identification of markers are very important for real time applications. For instance, directors can observe both the performance of the actor and the MoCap driven 3D model at the same time. Recent progress in computer vision led to the development of markerless optical systems. These systems do not require actors to wear special equipments. Specific algorithms are used to project the recordings from several cameras onto the virtual character. While these systems work effectively with large motions in real-time, they might not successfully capture small motions such as finger movements.

Inertial systems: Inertial motion capture systems are based on tiny inertial sensors. Inertial sensors capture the joint angles, and transmit this data to a processing unit wirelessly. The captured rotations are translated into a skeleton in software environment. These systems have low costs in terms of computation and finance, and unlike optical systems they do not require a specialized studio.

Mechanical systems: In mechanical systems a mechanical equipment that directly tracks the joint orientations is attached to a performer. Like inertial systems, they have low costs and uses wireless technology to transmit data. The ultimate drawback of such systems is that the special mechanical suit that captures the motion, limits the actors performance.

Magnetic systems: Magnetic systems use electro-magnetic theory for capturing position and orientation of each joint. Three orthogonal coils are placed on each transmitter and receiver. These systems acquire the motion data by measuring the relative intensity of the voltage or current. However they may be affected by the interference caused by electrical devices, and they are not highly reliable.

As the motion capturing systems became widespread, many industrial and academic research groups have dedicated themselves to improve the reliability and efficiency of these systems. Along with the developments in motion capture technology, various motion editing techniques, which aim to consummate the animation by altering the MoCap data, are proposed. Some of the important techniques will be explained at §3.1 in detail.

The main reasons for the existence of motion editing techniques are the limitations on the quality and the quantity of the captured data. Although the technology is widespread, capturing every single kind of motion is impossible. Thus, some altering techniques are proposed to modify the captured data in a way that it can be, at least partially, converted into another motion. Captured data might not be perfect in terms of quality and quantity. Hence, most of the time the raw data requires cleaning. Clean up is the process of correcting the errors on data that are generally caused by the capturing hardware. Other than clearing visually apparent errors, these methods improve the physical validity of motions. Other important reasons for motion editing are as follows:

- As mentioned earlier, one of the most common applications of motion capture technology is projecting the data onto another actor. However, if the

physical appearances of the actors are different, the output would not be realistic. Retargeting methods are employed for correcting such errors.

- Sometimes, especially in film industry, the captured data may not satisfy some criteria or the aimed scene may be impossible for a performer to act. Therefore, some visual effects may be required to alter the captured data.

The term locomotion has the dictionary meaning: 'the act of moving place to place'. In animation, locomotion refers to a group of basic daily motions, such as walking, running, and jogging in which the subject moves on ground. Since these motions are very common, they are frequently encountered in computer games and animations. Moreover, most of the main animation packages include locomotion generation functions. However, creating such motions with an arbitrary database is a challenging task, which requires selection of proper motions and applying motion specific adjustments.

In this research, we aim to develop a methodology for generating locomotion on real-time according to user-provided parameters. The methodology is also supported with an implementation. Another aspect of our study is to describe the preliminary steps to compose such a system; that is, what steps should be taken in order to select and create base motions from a large dataset. This procedure includes selection of motions according to predefined criteria, correction of errors on foot positioning, and pre-adjustments.

In order to create motions on real time, we constructed a graph based data structure. In this graph, each node represents a class of motions, and includes motions of that class with a variety of parameters. As in motion graphs [18], any walk on this graph can be converted into a set of sequential motions, which produces a large final motion. As an addition, our system provides the user the flexibility to change the motion parameters over the walk. Moreover, with our proposed methodology the system can easily be expanded, to include various other motions.

For transforming the walks and set of parameters into a long locomotion, inter and intra-blending schemes, which are based on *scattered data interpolation* and

linear interpolation, respectively, are described and implemented. The weighting algorithm in [33] is employed, for pre-computing the weight constants at off-line stage and calculating the parameter-specific weights for each motion on run-time.

The organization of the rest of the thesis is as follows. In Chapter 2, background on motion representation and motion figure types are provided. In Chapter 3, some of the distinguished works on motion editing are briefly introduced and discussed. A detailed explanation of our work is provided in Chapter 4. The implementation details and the evaluation of the results of our approach are provided in Chapter 5. Chapter 6 gives conclusions.

Chapter 2

Background

2.1 Motion Representation

In order to understand and apply motion reuse technologies, it is important to perceive how the poses and motions are represented. In this section, different types of motion and posture representations are explained.

2.1.1 Representing Poses

Motion capture data provides the pose of the character at each instant. This pose consists of values for all of the characters parameters at that instant. The choice of in how the poses are represented affects both the efficiency and effectiveness of a technique, as it provides actual numbers to be altered.

Typically, a hierarchical rigid skeleton is used to represent a character. Parameters of a skeleton consist of the position and absolute orientation of each piece (typically referred as bone) and relative orientations among connected pieces. In most of the editing techniques, local positioning of bones except the root is preferred, since it simplifies the required calculations. Because of this selection the rotation representation selection becomes more significant, since the local location

parameters are static in skeletons.

2.1.2 Representing Orientations

Orientation is expressed as a rotation relative to some other coordinate system, either a fixed 'world' coordinate system or another joint in the hierarchy. Indeed selection of how the orientations are represented is same as selecting another space to map from S^2. The spatial nature of this mapping affects the applicability of a reuse method.

2.1.2.1 Rotation Matrix

Rotation matrices are very useful in implementation step since most of the hardware and the libraries, such as OpenGL, support it. This representation provides a mapping from S^2 to $\mathbb{R}^9$. Some problems, such as difficulty in interpolation, arise because of the discontinuity of the inverse mapping function; however, some actions such as consecutive rotations can be performed very easily by simply multiplying matrices.

2.1.2.2 Euler Angles

Euler Angles represent the orientation as a fixed set of consecutive rotations around local coordinate system axes. Any rotation can be expressed as a rotation around the X axis, followed by a rotation around the Y axis and a rotation around the Z axis.

Euler Angles provide a better understanding of the rotation and a more compact representation than rotation matrix. However, they are prone to many problems. Since the mapping from S^2 to $\mathbb{R}^3$ is not continuous, the interpolation becomes problematic. Another problem, Gimbal Lock arises because three values representing the rotation are not independent. Despite these problems, Euler Angles are the most commonly used method for representing angles, since they do

not require sophisticated mathematical knowledge and are easy to observe with motion capture technologies.

2.1.2.3 Quaternions

The most common alternative to the Euler Angles for rotations is unit quaternions [32]. A quaternion consists of four values (x, y, z, w) , where x, y, z forms the vector part v, while w forms the scalar part s. A unit quaternion represents a rotation around vector v, of magnitude of s. A rotation of θ degrees around the unit vector u can be represented as:

$$q = \begin{pmatrix} s \\ v \end{pmatrix} = \begin{pmatrix} \cos(\theta/2) \\ \sin(\theta/2)u \end{pmatrix} \tag{2.1}$$

The most important advantage of the quaternions is their success in interpolation. For this reason, we used SLERP (Spherical Linear intERPolation) for quaternion interpolation in our implementation. Another advantage of quaternions is that successive rotations can be calculated by simply multiplying the quaternion. The details of calculating the exponential and logarithmic functions of quaternions are given below.

Given the quaternion q:

$$q = \begin{pmatrix} s \\ v \end{pmatrix} \tag{2.2}$$

The *exponential* of q is given as:

$$\exp(q) = \exp(s) \begin{pmatrix} \cos(|v|) \\ \frac{v}{|v|}\sin(|v|) \end{pmatrix} \tag{2.3}$$

and the *logarithm* of q is given as:

$$\log(q) = \begin{pmatrix} \log(|q|) \\ \frac{v}{|v|}\arccos(\frac{s}{|s|}) \end{pmatrix} \tag{2.4}$$

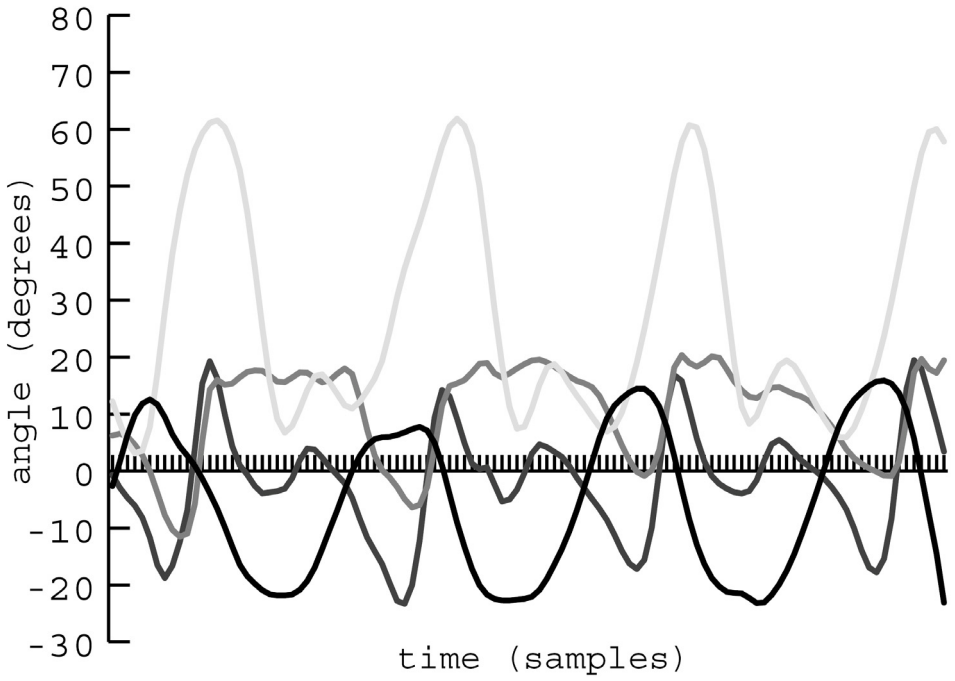


Figure 2.1: The angles of different joints on a small motion with respect to frames.

These functions will be used to map the quaternion displacements to $\mathbb{R}^3$ in posture blending.

2.1.3 Representing Motion

A motion is represented as a multidimensional function (Equation 2.5) that maps the time to a pose. It can be represented as a set of parameter curves, which corresponds to the position of root and joint orientations at time t (see Figure 2.1):

$$M(t) = (p(t), q_1(t), q_2(t), \ldots q_n(t)), \tag{2.5}$$

where $p(t)$ is root position and $q_i(t)$ is the orientation of joint i. It should be noted that each motion is defined as a set of frames, which means it is discrete and value of time t depends on frame rate.

2.2 Motion File Types

There are various types of files that use the representations above, such as Acclaim ASF/AMC, Biovision BHV/BHA, C3D, FBX. The most common ones used in motion editing are Acclaim ASF/AMC and Biovision BHV/BHA. In this section, some brief information about these file types are provided.

2.2.1 Biovision BHV/BHA

The BioVision Hierarchical (BVH) file format was developed by BioVision, a motion capture service company. The BVH format is a binary file that contains both a skeleton and motion capture data, and it allows each segment of a skeleton to have a specified order of transformation.

The BHV file format is very flexible and relatively easy to edit. It has two sections: The hierarchy section and the motion section. The hierarchy section contains the definition of a skeleton hierarchy within nested braces. The motion section of the file contains the total number of frames in the animation in the motion section, the frame rate, and the parameters for each entry (bone) in the hierarchy section. Figure 2.2 illustrates a BHV file.

A drawback of the BVH format is that it lacks a full definition of the initial pose. Moreover, the BVH format is often implemented differently in different applications.

2.2.2 Acclaim ASF/AMC

ASF (Acclaim Skeleton File) and AMC (Acclaim Motion Capture) are the file formats for MoCap developed by Acclaim Entertainment, Inc. The format includes two files: .ASF file, which describes the actual skeleton and its hierarchy, and the .AMC file, which contains the motion data. In Figure 2.3, you can see an example of each ASF and AMC files.

The ASF file contains all the information of the skeleton, such as bones, documentation, root bone information, bone definitions, degrees of freedom, limits, hierarchy definition, and file names of skin geometries, but not the motion data itself. In addition, the ASF file contains an initial pose for the skeleton.

The AMC file contains the actual motion data for the skeleton defined by an ASF file. The bone data is sequenced in the order as the order of transformation specified in the ASF file.

In our implementation, ASF/AMC files are used to store the motions. This choice was straightforward, since the motions in the database are initially stored in this format.

```
HIERARCHY
ROOT Hips
{
  OFFSET 39.5157 99.4896 24.7913
  CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
  JOINT ToSpine
  {
    OFFSET -0.0105055 1.38907 -7.13956
    CHANNELS 3 Zrotation Xrotation Yrotation
    JOINT Spine
    {
      OFFSET 0.0105055 10 1
      CHANNELS 3 Zrotation Xrotation Yrotation
      JOINT Spine1
      {
        OFFSET 0 12 1.60637
        CHANNELS 3 Zrotation Xrotation Yrotation
        JOINT Neck
        {
          OFFSET 0 27 2.26658
          CHANNELS 3 Zrotation Xrotation Yrotation
          JOINT Head
          {
            OFFSET 0 9 2.12705
            CHANNELS 3 Zrotation Xrotation Yrotation
            End Site
            {
              OFFSET 0 11 2
            }
          }
        }
        JOINT LeftShoulder
        {
          OFFSET 8 19.6109 2.86452
          CHANNELS 3 Zrotation Xrotation Yrotation
          JOINT LeftArm
          {
            OFFSET 12 1 0.681055
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT LeftForeArm
         . . .
 . . .
    }
  }
}                                # HIERARCHY PART ENDS HERE
MOTION
Frames: 153
Frame Time: 0.0333333
39.5107 99.4851 24.7919 1.29283 -5.72371 0.672317 0 0 0 2.52727 5.05271
0.319096 -5.56434 3.32673 1.29694 8.53147 3.12242 -2.34842 -4.43757
-2.29464 1.341 -29.1055 3.86484 -7.45312 -41.6174 27.5825 -18.4723
-6.84645 -13.417 0.262612 -9.01332 0.37924 -4.80751 26.2911 6.48437
16.3051 50.6802 17.8657 5.35063 5.04918 -6.43848 9.88527 2.839 0.0844801
3.41222 0.52226 7.09518 7.26472 -1.1878 5.46476 -9.93351 3.56859 -7.9309
0.247298 0.693993 2.24148 -3.2565 -3.21779 7.68437 -9.34312 0.0551059
5.01404 -4.71352 -8.88908 -3.36204 -0.261081 3.329 -12.1909 -3.65467
. . .
```

Figure 2.2: An example of BHV file for crawling motion.

```
# AST/ASF file generated by VICON BodyLanguage
# ------------------------------------------
:version 1.10
:name VICON
:units
  mass 1.0
  length 0.45
  angle deg
:root
   order TX TY TZ RX RY RZ
   axis XYZ
   position 0 0 0
   orientation 0 0 0
:bonedata
  begin
     id 1
     name lhipjoint
     direction 0.655637 -0.713449 0.247245
     length 2.52691
     axis 0 0 0  XYZ
  end
  begin
     id 2
     name lfemur
     direction 0.34202 -0.939693 0
     length 7.59371
     axis 0 0 20  XYZ
    dof rx ry rz
    limits (-160.0 20.0)
           (-70.0 70.0)
           (-60.0 70.0)
  end
...
:hierarchy
  begin
    root lhipjoint rhipjoint lowerback
    lhipjoint lfemur
    lfemur ltibia
    ltibia lfoot
    lfoot ltoes
    ...
  end
```

```
#!OML:ASF TakeoMonday.ASF
:FULLY-SPECIFIED
:DEGREES
1
root 9.62745 17.7973 -1.03923
    -6.37909 -22.4014 3.49382
lowerback 5.60025 1.12798 -0.0431347
upperback 2.77139 1.49305 0.791281
thorax -0.311801 0.736113 0.749946
lowerneck -8.90502 6.57112 0.322245
upperneck 1.33699 8.87524 -3.9415
head 2.15388 4.30658 -0.892614
rclavicle -5.42682e-014 8.74653e-015
rhumerus -26.2308 10.8446 -85.4595
rradius 32.4455
rwrist -15.5297
rhand -23.7652 16.4224
rfingers 7.12502
rthumb 2.70415 -13.5402
lclavicle -5.42682e-014 8.74653e-015
lhumerus -31.3304 15.8471 84.3404
lradius 34.38
lwrist 26.2658
lhand -24.6861 18.635
lfingers 7.12502
lthumb 1.81469 48.607
rfemur 5.94355 -4.3449 19.6053
rtibia 16.8854
rfoot -16.8444 -20.544
rtoes 1.44864
lfemur -6.98547 0.557427 -21.3734
ltibia 17.0463
lfoot -2.75734 18.7084
ltoes -9.6083
2
root 9.62656 17.8064 -1.04088
    -6.77019 -22.4714 4.31411
lowerback 5.63075 1.18462 -0.822496
upperback 2.76579 1.52792 0.503344
thorax -0.347203 0.748779 0.888507
...
```

Figure 2.3: An example of AMC/ASF file pair for walking motion.

Chapter 3

Related Work

With the developments in motion capturing technology, motion editing became a popular subject. In last fifteen years, there have been many studies that contributed the progress in this area. In this section, we provide a compilation and analysis of the most significant works in this subject.

3.1 Motion Synthesis and Editing Techniques

In the field of motion editing, many algorithms and techniques for generating realistic motions from captured motions are proposed. These methods can be classified into categories according to their perception of the problem and the approaches they provide. These categories are explained in detail below along with some of the most important studies in each one.

3.1.1 Manual Synthesis

Manual synthesis is the earliest form of motion synthesis. It is actually the classical cartoon drawing technique adapted to computer environment with some

simple interface and in-betweening techniques. Burtnyk et al. [8] describes basic adaptation schemes and interpolation techniques for keyframed animation. In [20], John Lasseter describes the simple concepts for traditional animation that can be applied to keyframed animation for visually better results. As the animator specifies individual DOFs (Degree of Freedom) and joint torques at all keyframes, he will have full control over motion. However, introducing so many keyframes in the absence of appropriate automation techniques is a tedious work. Therefore, several methods that aim to shoulder this load as much as possible are introduced in literature.

3.1.2 Forward/Inverse Kinematics

As mentioned earlier, most of the motion reuse techniques define the configuration space in $SO(n)$ where n is number of joints. However, the tasks and the environment are described with respect to workspace $\mathbb{R}^3$. In other words, the joint parameters are represented as local orientations, but the constraints such as foot positioning and specific joint isolations are represented in Cartesian space world coordinates. This difference in representations forces invocation of forward and inverse mappings. These mapping functions are named Forward Kinematics (FK) and Inverse Kinematics (IK), respectively [4].

Let the vector $\vec{\Theta}$ be the skeleton's orientation and $\vec{P}$ be its respective world coordinate in Cartesian space. The sizes of the vectors $\vec{\Theta}$ and $\vec{P}$ are equal to the structure's DOF. Then Forward Kinematics mapping is described by f, if:

$$\vec{P} = f(\vec{\Theta}) \tag{3.1}$$

Straightforwardly, f^{-1} defines Inverse Kinematics, that is:

$$\vec{\Theta} = f^{-1}(\vec{P}) \tag{3.2}$$

Until recent years, published works focused on describing an efficient way for solving and applying inverse kinematics in computer animation. Various approaches are proposed for this problem. These methods can be categorized into

three groups: *analytical*, *iterative*, and *hybrid*. Analytical methods [9, 31, 34, 44] define a set of algebraic equations and then solve them, while iterative methods use Newton-Raphson root finding method to solve IK problems. In Cyclic Coordinate Descent (CCD) [37] method, an iterative approach updates the joint angles until the target destination is reached. This method is popular since it has the speed of an analytical approach and effectiveness of an iterative approach. The approach proposed in [21] uses hybrid systems that solve the minimization problem for reduced set of bones.

The main usage of inverse kinematics methods in animation is to force the global constraints during or after motion synthesis; that is, finding the character poses that satisfies the given constraints. In literature, many solutions for this problem are proposed [5, 12, 38]. However, as one can imagine, this problem has more than solution including the ones that are not physically correct or visually satisfactory. Due to this undetermined nature of the problem, another problem arouses: selecting the most appropriate posture among many possible solutions. Because of the difficulty of this selection process, the role of inverse kinematics is limited to supplying correction algorithms for motion synthesis in most of the proposed systems, such as [24, 28, 45, 46].

There are exceptional researches that directly create motions/poses using Inverse Kinematics, or narrow the solution space in order to increase the possibility of finding visually/physically sufficient ones. In [26], mass displacement from a reference pose is used as a measure for correctness of generated motion. Similarly, Grassia calculated this metric by measuring the energy consumption of a motion [13]. Popović et al. applied training algorithms to minimize distance of a pose to the ones in the training examples [14]. They tested their algorithms on applications such as interactive character posing, trajectory keyframing, real time motion capture with missing markers, etc. and the approach seemed effective. In one of their demonstrations, they showed that the system is capable of creating poses that matches a real pose of a baseball player, with only adjusting a few end-effectors consecutively.

3.1.3 Physically-based Synthesis

As the human motions are affected and sometimes completely controlled by physical laws, several motion synthesis approaches are proposed that keeps the motion in the boundary of fundamental physic laws, such as Newton's Laws. This kind of approach requires information more than joint angles and positions, such as mass distribution and the joint torques. The biomechanics, including mass distribution information, is well explained in [40]. Hodgins et al. [16] focused on joint torque calculations for highly dynamic motions such as running, vaulting and bicycling. They used finite state machines to enforce a correspondence between the phase of the behavior and the active control laws; and used proportional derivative control laws for low level control. In [42, 43], similar methods, for generating gymnastic motions are introduced.

These methods, in general, formulate the synthesis as optimization functions, whose constraints are based on given task and a selected set of applicant physics laws [11]. In [41], space-time constraints are introduced to describe how the motion should be performed inside the boundaries of physical validity. The method proposed in [6] used search algorithms for torque/force generators (controllers). Hodgins and Pollard [15] additionally adapted a controller to a new body.

Hybrid methods that use biomechanics data have also been studied. The studies [1, 26] proposed different methods for modifying input motion to obtain new motions with different objectives. Liu et al. [22] also invoked a hybrid method. They tried to optimize minimum mass displacement of a reduced model, in light of physical constraints such as momentum and different types foot/hand contacts. Their system was able to handle highly dynamic motions, such as running, hopscotch, high bar and handspring. Popović et al. [10] also proposed a hybrid method. Their approach determines controllers in the guidance of style and balance feedbacks from reference motions. They used iterative optimization on a reduced model, a three link model, for computing the corresponding control forces of each reference motion.

Physical approach sometimes generates motions that lack personality. Ne and

Fiume [23] tried to overcome this kind of stiffness on such motions by focusing of tension and relaxation. Safonova et al. [30] reduced the dimensionality, DOF, of the motion using Principal Components Analysis (PCA), then applied physically constrained optimization on this model and increased the model dimensions back to normal. By solving the optimization for higher levels, the unchanged lower levels help maintaining the personality of the motion.

Physical-based methods combined with data driven techniques are also expanded to produce dynamic responses for outer forces. Arikan et al. [3] synthesized response and balancing motions for a wandering character after a push in different directions. Zordan et al. diversified the falling motion of a human body after a short and strong impact, such as being kicked, from a few captured examples [46].

In summary, physically based methods typically generate realistic body configurations and motions; however, the animator should be aware of the stiffness, which is brought by most of these techniques.

3.1.4 Data-Driven Synthesis

Motion capture technologies provide reliable realistic motions. As this technology become widespread and used to gather large sample sets, data-driven methods became applicable for creating new motions. Blending techniques (see §3.1.4.1) are also considered as data-driven methods.

In some studies, the motion data is treated as a set of signals. Signal processing methods are applied to these signals to alter the captured motion. Unuma et al. [35] introduced this approach; they extrapolate and interpolate the Fourier coefficients of joints between walking motions of different moods, in the frequency domain. In [7], motions are also processed in frequency domain. They provide the system user a graphical equalizer of gains on frequency bands of joint angles. With this technique, the user can generate anticipation effects with tedious effort, because the correspondence between the parameters on equalizer

and output motion is not good. In other words, it was hard to anticipate what kind of effects it will create on the output motion, after changing parameters. Wang et al. [36] used inverted Laplacian of a Gaussian (LoG) filter to create anticipation and follow through effects as described earlier by John Lasseter [20]. Their work greatly extended previous techniques. With the one parameter interface, the user can specify the exaggeration magnitude easily. Their unified approach was applicable not only to MoCap data but also to video recordings, and even, on simple animations on PowerPoint.

3.1.4.1 Motion Blending

In this technique, a set of similar motions are blended. The blending is typically achieved by interpolating the basic parameters, such as joint angles and root positions. Ken Perlin's work [25] was one of the first studies that include motion interpolation. In this system, blending operations are applied on a motion dataset to create new motions and transitions between them. Wiley and Hahn [39] used linear interpolation and spherical linear interpolation on a set motions including pointing and reaching behaviors to create new directions for these actions.

In [27], radial basis functions are invoked for interpolating locomotions. Rose and his colleagues defined some analogous structures for simplicity; the base example motions are referred as verbs, while the control parameters describing these motions, such as mood, are called adverbs. The overview of their system is illustrated in Figure 3.1. As a result of their research they succeeded in creating new motions by interpolating example motions with new values for adverbs. In our work, we have used an incremental version of their approach for interpolation.

In most of the interpolating approaches proposed, linear interpolation is used widely for position values and spherical linear interpolation for quaternion representations of joint orientations.

A similarity metric for frames of two motions is proposed in [17, 18]. In these works, they used this metric to find appropriate interpolation timings in between the motions. Their system generates motion graphs using these times for creating

edges/transitions. They also described a technique for automatically registering motions for interpolation.

Shin et al. [24] use interpolation techniques for creating an on-line locomotion motion with given parameters. In their work, they manually clip some short segments of the motions with the exact keyframe sequence and use time warping for synchronizing these motions before interpolating them according to the given set of control parameters, as in [27]. Shin and Kwon [19] developed a system for automatic segmentation and classification of long motions for using them in a similar system.

Safonova and Hodgins [28] analyzed the interpolated motions that have static control parameters for flight phase (with no ground contact) with respect to physical validity. They tested the angular and linear momentum of created motions, along with stability of foot during contact and static balance. They showed that, in many cases, the interpolation method creates physically valid motions.

Recently, Safonova and Hodgins [29] constructed motion graphs with nodes, that have two similar poses to be interpolated and a weight value. This representation increases the flexibility of generated motions, while maintaining graph structure. By doing so, long sequence motions can be created by simply following transitions on the graph, as in original paper [18]. A search algorithm that optimizes the weight values at each node according to given sketch and tasks is also provided.

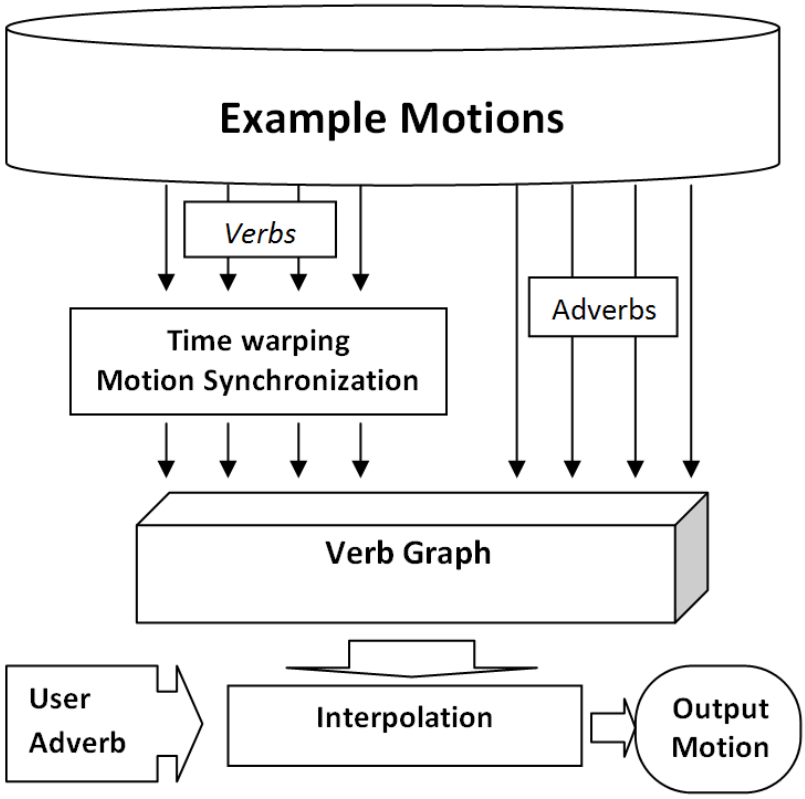


Figure 3.1: The overview of the system described in [27].

Chapter 4

Real-Time Locomotion Generation

In this work, we aimed a system that is capable of generating locomotion, such as walking and running, with user-specified parameters. The general overview of our system is shown in Figure 4.1. The system consists of two stages. The first stage is the motion graph construction. The motions are selected based on some criteria and error correction, pre-adjustments, motion snychronization, and transition partitioning steps are applied to construct a motion graph from a huge database of raw motions. The second stage uses the motion graph constructed in the first stage and converts the nodes of the graph into motion segments using data scattered interpolation. Then, an output motion is generated by concatenating these motion segments. The concatenation is done by linearly blending the motion segments with respective transition motions, which are represented as edges on the graph. The main structure of the motion graph is constructed at off-line stage, while the second part, motion generation, is done during runtime. This separated manner in system flow, improves the overall functioning and efficiency of the system.

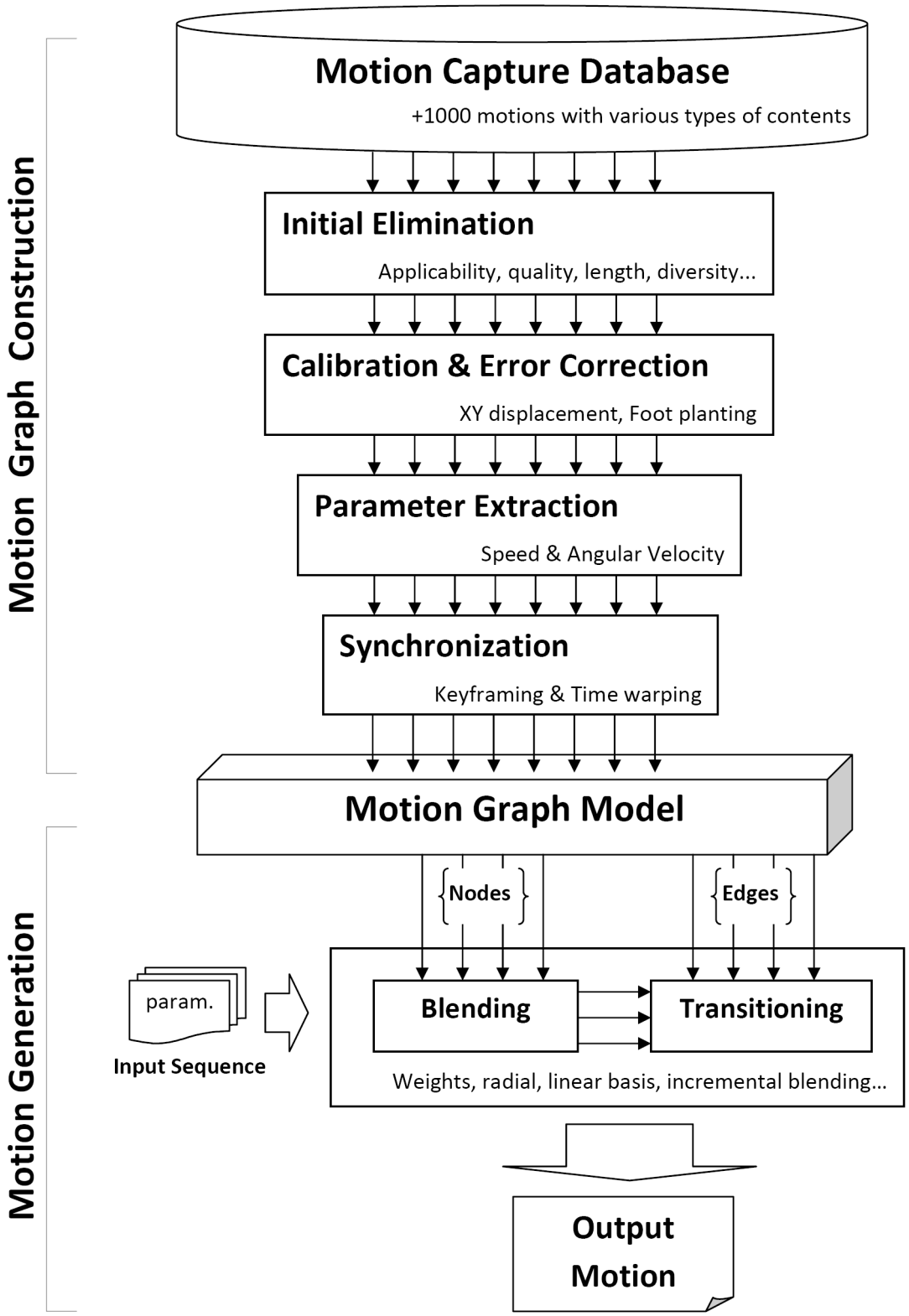


Figure 4.1: The overview of the real-time locomotion generating system.

4.1 Graph Construction

For generating locomotions, with flexibility of anytime transitioning, we used graph structures as in [18, 19]. In Figure 4.2, a reduced visual model of our graph is presented. Unlike motion graphs [18], each node at our graph represents a set of similar motions, such as running. The set of motions at each node are first selected from a huge database of motions, and labeled according to their action content. Then, the visible errors on these motions are eliminated by Linear Function Fitting. After the selected motions are calibrated, keyframe timings are synchronized using Incremental Time Warping according to the other motions at each node.

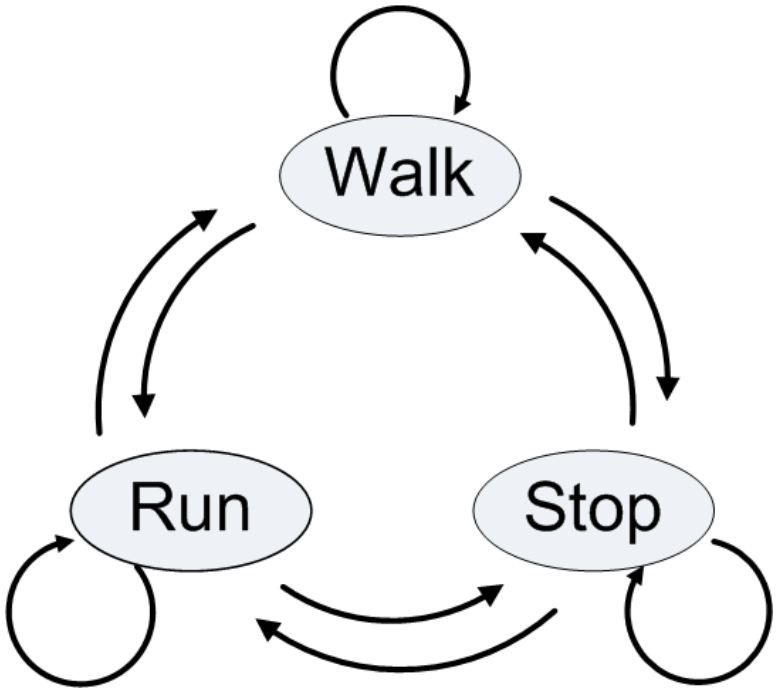


Figure 4.2: The motion graph model of our system.

Each edge in our graph model represents a transition between motions. Example transition motions between every single pair of nodes are selected in the same manner and pruned as in Motion Graphs, in order to cut off redundant frames. After these steps, any walk in the graph can be converted in to a long motion by sequencing the blended sets of motions at each graph one after another. This step will be explained in §4.2.

4.1.1 Selection of Example Motions

For successful transitions and blending of motions, the quality and diversity of the example motions are very crucial. In our work, we selected our motions from MoCap database of Carnegie Mellon University, which includes large amounts of motions of varying styles, actions and qualities. This diversity of the database imposes the problem of selecting right motions. In choosing appropriate motions for our implementation, we take into account the following criteria:

- *Motion action content and styles:* The first criterion should obviously be the content of motions. Fortunately, the on-line database provided labeling of most of the content. According to these labels, we extracted hundreds of motions for each category that matches this criterion.
- *Number of frames and frame rate:* The motions are expected to be long enough to be blended into meaningful and smooth motions. The motions, whose number of frames is lower than a threshold, are removed from set of candidates. Motions with dissimilar frame rates are also omitted for compatibility issues.
- *Predicted motion parameters:* In order to predict the motion parameters that will be precisely extracted after the selection, we implemented small scripts that identify overall change and average of the parameters. Unfortunately, the database included some repetitive motions that would be eliminated by our filter, although they have partially acceptable motions. Hence, we applied our script to motion segments of predetermined sizes.
- *Diversity of candidate motions:* After eliminating undesired motions, we grouped the motions according to their predicted parameters.
- *Motion quality:* Capturing motions in large batches reduces the quality of MoCap data. Although correction algorithms are implemented for eliminating such errors (see §4.1.2), it cannot recover all of the errors, such as joint angle registration flaws and disconnectivity in the motions. Therefore,

we eliminated motions at each group with lower quality, and nominated one motion from each parameter group.

The nominated members of each group of the same action content are gathered into a node model.

4.1.2 Motion Error Pre-correction

Correction of motions consists of aligning the starting position and orientation of first poses according to a reference point and direction. For calculative simplicity, we selected the reference position as the origin of X-Y plane of global coordinate systems and X-Axis direction is selected as reference direction. The positional displacement vector $\vec{d}$ is formulated as:

$$\vec{d} = -\vec{p_1}, \tag{4.1}$$

where $\vec{p_1}$ is the root position of the first pose in the motion. The angular displacement θ can be found by calculating the angle between X-Axis and the tangent of the arc formed by the relative root positions in first n frames.

Error pre-correction plays a major role in our approach. It eliminates the footing errors at the stage of graph building, unlike many similar methods that use Inverse Kinematics after the motions are generated. This choice is grounded on works of Safonova and Hodgins [28]. In their research, they have shown that the physical validity of motions to be interpolated is directly reflected on correctness of interpolated motion. By correcting the errors in advance, we aim to reduce the load on CPU at motion generation stage, while preserving the physical validity. In the pre-correction step, the motion is repositioned as close as possible to the ground. This step is required since the capturing technology used was not capable of precisely aligning motions into the world coordinates. The motions used in our implementation had a visible amount of deviation from ground in general, which increases or decreases linearly with the internal time of the motions. In order to solve this issue, each motion is considered as a rigid body consisting of poses with

static links among them, like a bulk iron statue. We tried to fit the minimum points, that are actual ground contact points of this rigid body into a line L formulated as $y = ax + b$. For this purpose, least square method is used to solve the error function for a and b:

$$\min_{a,b}(\sum_{i=1}^{n}(ax_i + b)^2), \tag{4.2}$$

where x_i is the foot position of ith minimum point of the motion and is calculated using Forward Kinematics. By interpolating a and b values for other frames, we found a y-displacement for each frame that will minimize the overall distance of the motion to the ground.

4.1.3 Parameter Extraction

In most of the studies, locomotion parameterization is based on three components: speed, turning angle, and style [19, 30]. In this work, we stick to this approach. However, due to the limitations caused by the diversity of parameters at style context, we narrowed the parameter space to contain only speed and angle. As mentioned earlier, our example motion data set have three types of motions: walk, run and stop. In the sequel, we will describe the formulations for extracting the parameters of walking and running. The parameters of the stop motion are assigned to zero, due to its immobile nature.

We calculate the angular velocity (ω) and linear velocity (v) parameters, as discussed in [24]. According to classical physics, an object with constant linear and angular velocities follows a circular trajectory. Grounding this fact, we calculated a best fit of motions into circular arcs of finite length. It should be noted that no classification of trajectories into 'turning' and 'straight' locomotions is required, since a straight line can be expressed as an arc of infinite radius.

Let $\tilde{p}$ be the projection of the root trajectory of a motion on the floor, we approximate $\tilde{p}$ as a circular arc a of radius r, centered at o. The arc also has

starting point a_0 and subtend angle θ. The least square fitting of this minimization is formulated as:

$$\min_{o,a_0,\theta} \sum_{i=1}^{F} (\tilde{p}_i - p_a(i; a_0, o, \theta))^2, \tag{4.3}$$

where F is number of frames, and $p_a(i; a_0, o, \theta)$ is the starting position of ith segment of the arc defined by a_0, o and θ, which is split into F segments. Then, the speed and the angular speed of the motion are given by:

$$v = \frac{r\theta}{T} \ , \ \omega = \frac{\theta}{T}, \tag{4.4}$$

where T is the duration of motion. It should be noted that at this step, v should be calculated by simply averaging the root positions for motions with infinite radius.

4.1.4 Weight Computation

We define the weight functions according to the scattered data interpolation method, as described in [33]. In the graph building stage, we use the weight functions and the parameter vectors to calculate the constant matrices for weight computation. The details of using weights for interpolation will be discussed at §4.2.3. Let p be our parameter vector. Then, the weight $w_i(p)$ of example motion i is defined as:

$$w_i(p) = \sum_{k=0}^{N_p} a_{ik} A_k(p) + \sum_{l=1}^{N_e} r_{il} R_l(p), \tag{4.5}$$

where N_p is the number of parameters, N_e is number of example motions, R and r are the radial basis function and its N_exN_e coefficient matrix, and A and a is the linear basis function and its N_exN_p coefficient matrix, respectively. $R_l(p)$ is the radial basis function of the Euclidean distance of p and parameter vector of example motion l, p_l:

$$R_l(p) = B\left(\frac{\|p - p_l\|}{\alpha_l}\right) \ , \ \ l \in [1, N_e], \tag{4.6}$$

where α is the dilation factor and $\alpha_l = min(p_j - p_l)$, for $j, l \in [1, N_e]$, and B is the cubic spline. Linear basis function $A_k(p)$ is equal to p_k for $k > 0$, and 1 if $k = 0$.

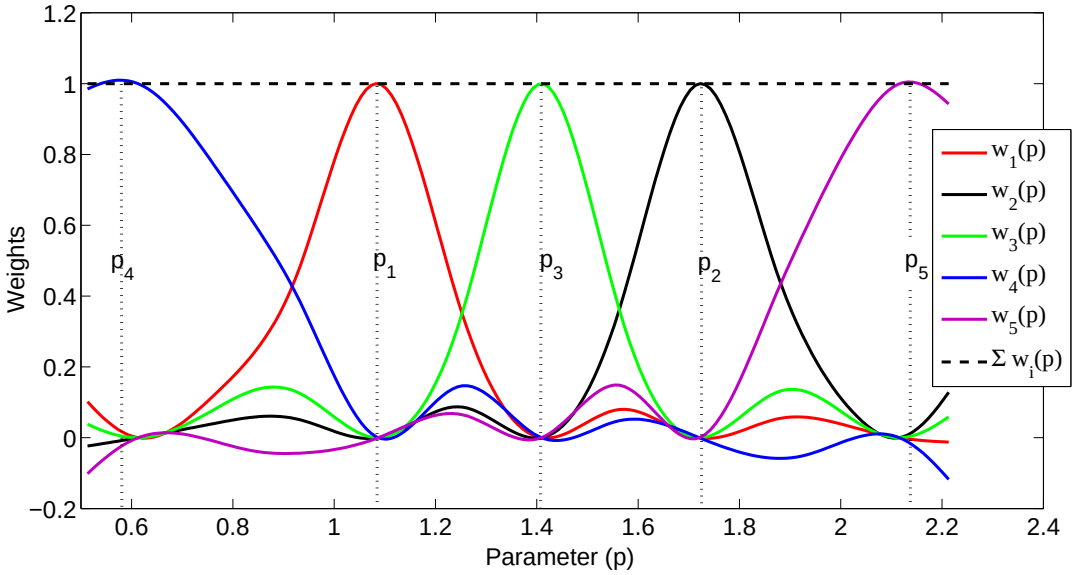


Figure 4.3: The weight values according to the user-specified parameters.

Given the weight formulations and parameter vectors of example motions, the coefficient matrices r and a is calculated by assigning weight values for parameter vector p_j of each example motion j for $1 \leq j \leq N_e$:

$$w_i(p_j) = \begin{cases} 1, & if\ i = j \\ 0, & else \end{cases} \tag{4.7}$$

First, matrix a is calculated by employing a least square method for the first part of the weight equation (omitting the last part of the formula):

$$w_i(p) = \sum_{k=0}^{N_p} a_{ik} A_k(p). \tag{4.8}$$

Having a values, r matrix is calculated by solving the linear systems given in Equation 4.9. Obtaining a and r values, we can calculate weight value for an arbitrary vector p by using Equation 4.5, and forward it as an input for our

blending scheme.

$$rR = w_i(p) - \sum_{k=0}^{N_p} a_{ik} A_k(p), \tag{4.9}$$

where $R_{ij} = R_i(p_j)$.

4.1.5 Motion Synchronization

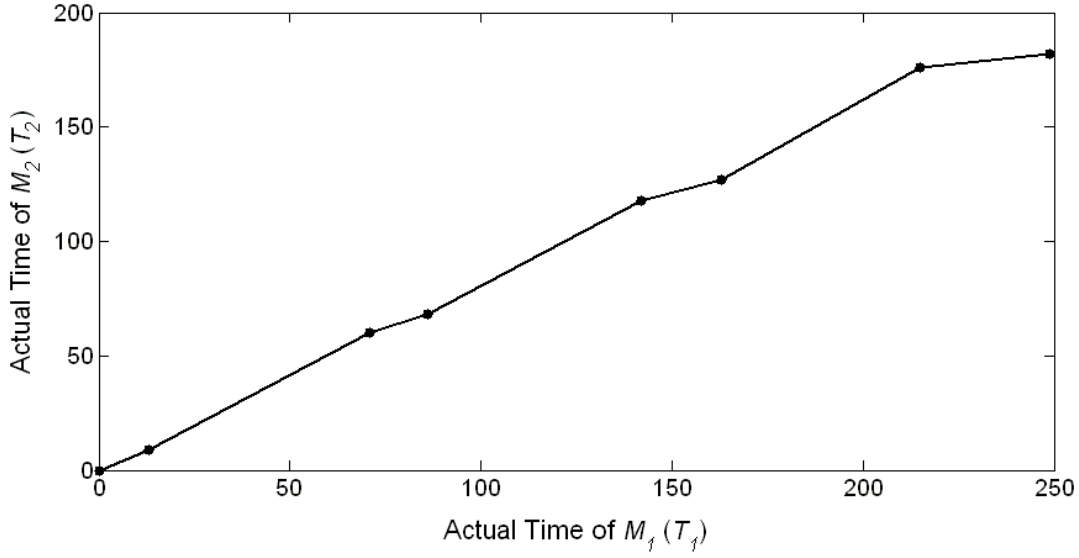
In order to generate longer motions with heterogeneous speed and angular velocity, we need compatible small scaled motions with nearly stable parameters. In this section, we find the key frames of motions, prune them respectively and synchronize them using incremental time warping.

4.1.5.1 Keyframing

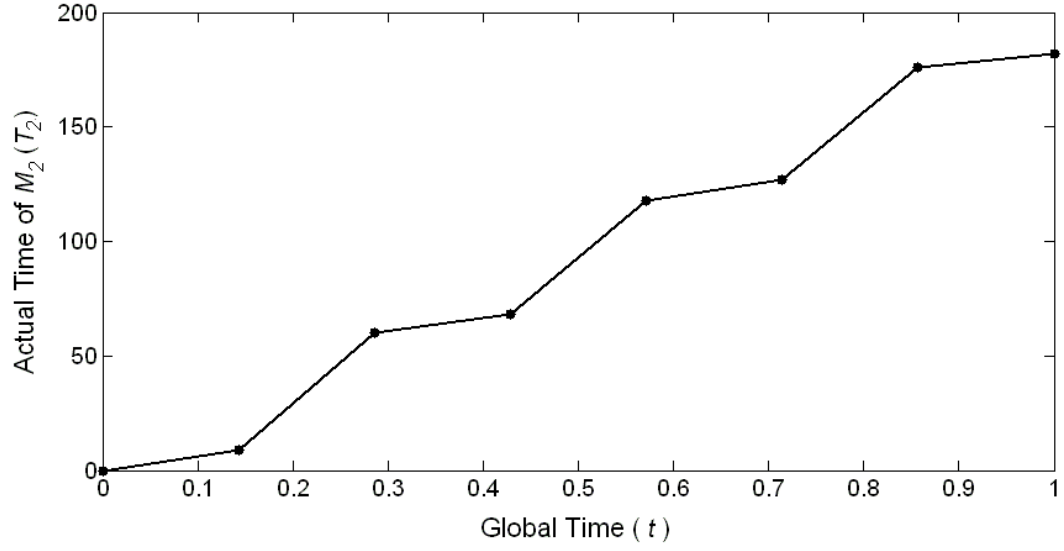
Keyframes are the important instants of a motion, since they are frequently used to describe the motion roughly. In locomotions, the significant frames are the ones with foot contact. Therefore, we defined the keyframes of our scheme to be the beginning and end frames of each foot contact. The extraction of keyframes is done manually. Yet, we developed a method for easier analysis. The height of heels and toes are plotted and the patterns for keyframes are observed. According to these observations, the keyframes are successfully labeled on the plot, avoiding the process of playing back and forward the motions each time

4.1.5.2 Time Warping

Time warping plays an important role in motion blending, since it formulates the correspondence between frames of motions that should be interpolated. Having the keyframes, a linear mapping from frames of a motion M_i to motion M_j can be calculated effortlessly. However, this pair wise mapping would bend the space of M_i, causing loss of realism. Moreover, for blending more than two motions this scheme does not work. Therefore, it is required to define a global time along



(a)



(b)

Figure 4.4: A comparison of pair wise linear mapping and global linear mapping: (a) linear mapping between two walking motions M_1 and M_2, (b) linear mapping between actual and global time of M_2.

with its mapping functions for each motion. We employed the incremental time-warping scheme described in [24]. The global time is calculated by distributing the keyframes to $[0, 1]$ interval uniformly. Given keyframes $[K_1, K_{N_k}]$ of motion M_i for $1 \leq i \leq N_e$, where N_e is number of example motions and N_k is the number of keyframes for that motion, actual time T_i can be mapped to global time $t(T_i)$ as follows:

$$t(T_i) = \left((m-1) + \frac{T_i - K_m}{K_{m+1}} - K_m \right) \frac{1}{N_k - 1}, \tag{4.10}$$

where m is the largest index such that $t(T_i) > K_m$. As seen in example in Figure 4.4 the mapping between actual time and global time is monotone, therefore the inverse function $T_i(t)$ exists, and defined as:

$$T_i(t) = ((N_k - 1)t - (m-1))\ (K_{m+1} - K_m) + K_m. \tag{4.11}$$

In Figure 4.5, the before and after effects of timewarping is illustrated. Formulating the forward and inverse mappings between actual times and global time, we are able to find the corresponding frames of motions in a node, given global time t. The update function for t is defined in §4.2.3. With this step, the graph construction stage is completed.

4.2 Motion Generation

In this section, we will define the algorithms and formulations required to efficiently blend these motions and create transitions on run-time.

4.2.1 Overview

Given the constructed motion graph, a sequence of parameters, first, converted into a graph walk according to specified motion types. Then, the graph walk is converted into output motion step by step:

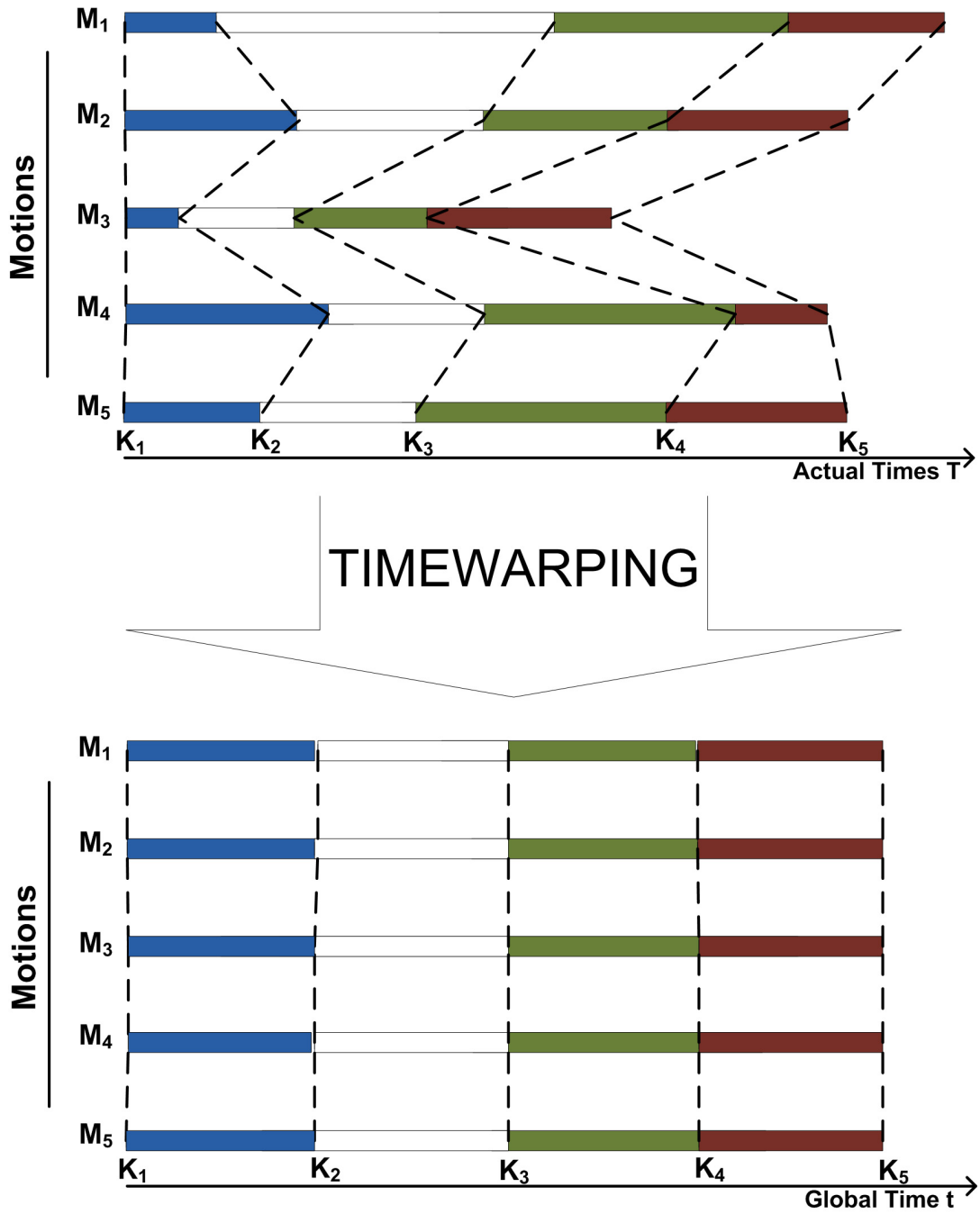


Figure 4.5: The effect of timewarping on motions.

- A local timing scheme is attached to each node for calculating the corresponding frames of each motion inside. According to given parameter sequence and length for each motion type, the motions in the corresponding nodes are blended, and one output motion for each node is generated. While blending, the same approach in §4.1.4 is used to compute weights of each example.
- The frames between the consecutive nodes are created using predefined transition motions on edges between those nodes. The transition motion and the output motion of the following node are transformed according to global position and orientation of the model.
- The transition motions are partially blended with the output motions of preceding and succeeding nodes using linear interpolation in order to create smooth transitions.
- The transformed and blended motions are concatenated one after another in order to form the final output motion.

4.2.2 Sub-global Timing

As mentioned earlier, for synchronizing motions in a node a global timing scheme is required. However, this scheme is used only inside the motions; therefore, it will be referred as sub-global timing scheme. We have already defined the nature of this scheme and the maps between the sub-global time and local times of the motions in §4.1.5. In this section, the initialization and update function of the sub-global time will be explained.

Let t_n^i be the time at the n^{th} frame of output motion of node N_i. The initialization step is quite simple, that is, t_n^i=0. For calculating the time of next frame an incremental approach is employed, and t_n^i is formulated as follows:

$$t_n^i = t_{n-1}^i + \Delta t_{n-1}^i. \tag{4.12}$$

In order to preserve the original frame rate of each motion, Δt is calculated

by interpolating the sub-global time change per frame,$\Gamma_j(t)$ for each motion j.

$$\Delta t^i_{n-1} = \left(\sum_{j=1}^{N_e} \gamma_j(p)\ \Gamma_j(t^i_n - 1) \right), \tag{4.13}$$

where N_e is number of example motions in the node. γ_j is the weight for motion j according to given parameter p, and is formulated as follows:

$$\gamma_j(p) = \frac{1}{N_e} + \sum_{k=1}^{N_e} r_{jk} R_{jk}(p). \tag{4.14}$$

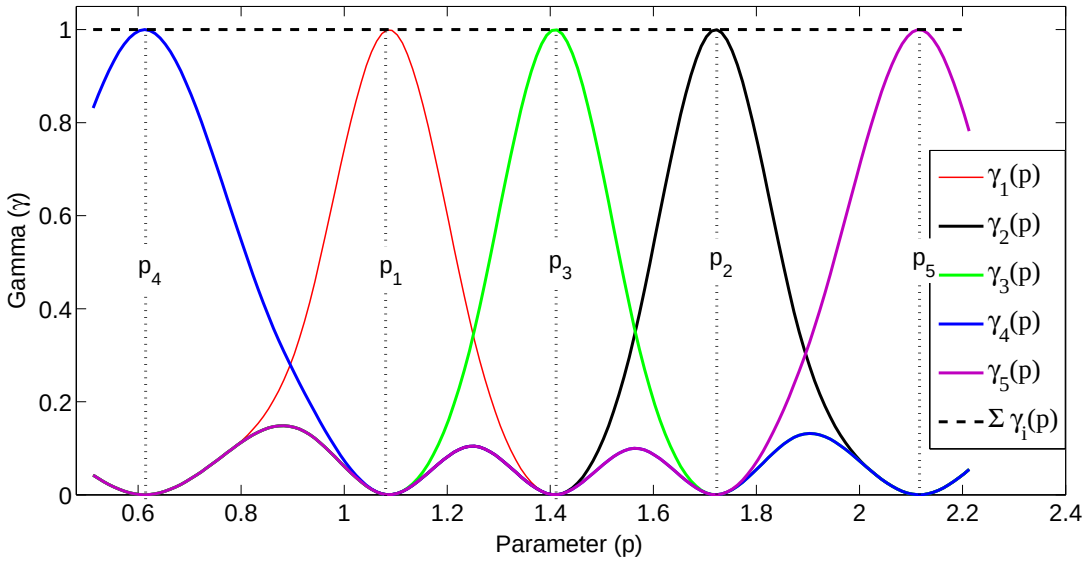


Figure 4.6: The γ values for the user-specified parameters.

Here, R is the radial basis function, as formulated in Equation 4.6 and shown in Figure 4.7. r is $N_e \times N_e$ coefficient matrix, and it is calculated by solving the linear equation:

$$rR = \gamma_j(p) - \frac{1}{N_e}. \tag{4.15}$$

The constant $\frac{1}{N_e}$ plays a critical role, for calculating Δt. It simply ensures that the weight, γ, for each motion and Δt is non-negative. The updates continues until the sub-global time reaches 1.0 limit.

With the timing scheme described, the corresponding posture of each frame can be found using the mappings defined earlier. These postures are blended

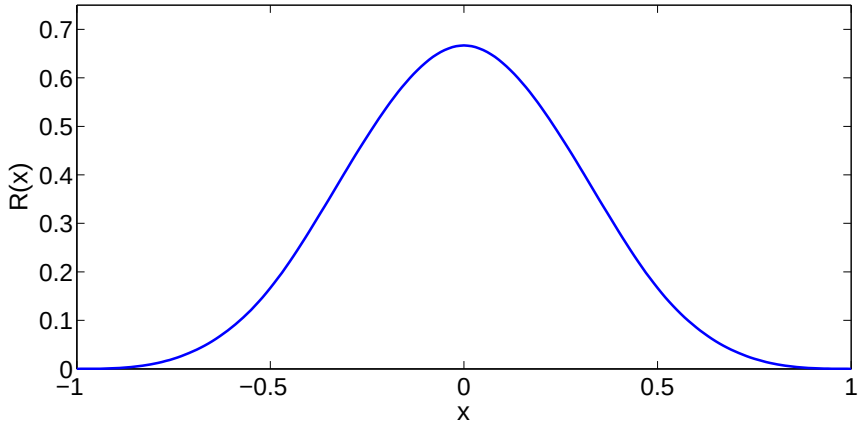


Figure 4.7: The radial basis function.

based on a data scattering interpolation approach, which is explained at the following section.

4.2.3 Incremental Posture Blending

By blending the corresponding postures of example motions at generic time t according to weight values for target parameter vector, the target posture is generated. As mentioned earlier, a posture $P(t)$ can be represented as:

$$P(t) = \left\{ p_r(t), q_1(t), q_2(t), \ldots, q_{N_j}(t) \right\}, \tag{4.16}$$

where N_j is number of joints in our skeleton model. In our implementation, we used one of the skeletons defined in the MoCap database; the joint hierarchy of our model can be seen at Figure 4.8.

4.2.3.1 Incremental Position Blending

For interpolating the root positions of the corresponding frames at generic time t_n, we use an incremental method. In this method, we interpolate the positional

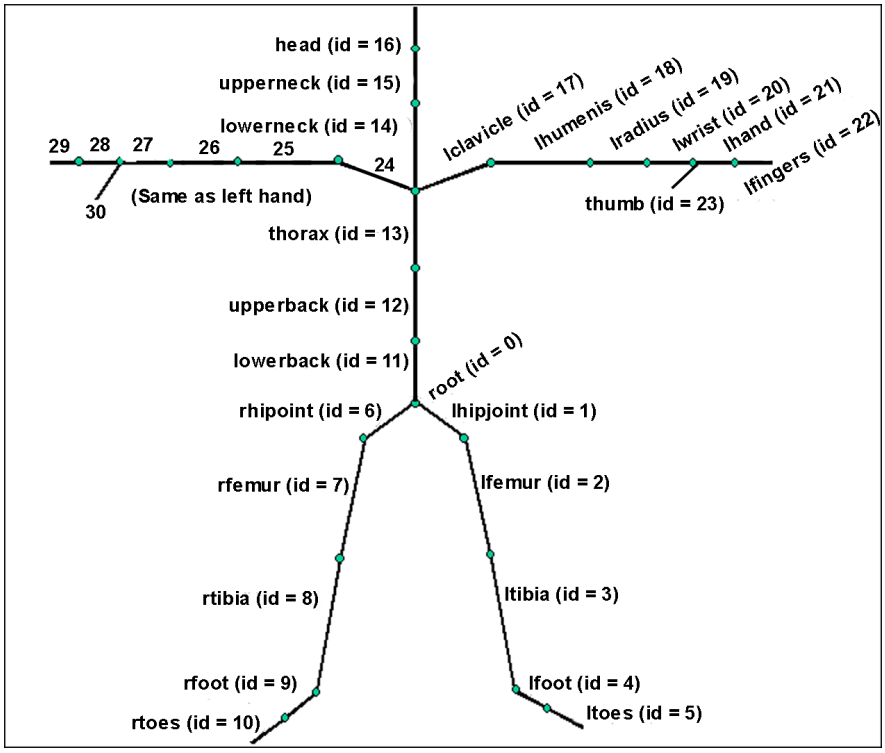


Figure 4.8: Our skeletal model.

displacement of corresponding frames. Let $p_i(T(t_n))$ be the root position of posture $P_i(T(t_n))$ for actual time $T(t_n)$, then the displacement $\Delta p_i(T(t_n))$ is:

$$\Delta p_i(T(t_n)) = \begin{cases} \vec{0}\,, & if \quad T(t_n) = 1 \\ p_i(T(t_n)) - p_i(T(t_n) - 1), & else \end{cases} \tag{4.17}$$

The root position p_G of output posture at $T_G(t_n)$ is calculated as:

$$p_G(T_G(t_n)) = \sum_{i=1}^{N_e} w_i \Delta p_i(T_i(t_n)) + p_G(T_G(t_{n-1})), \tag{4.18}$$

where w_i is the weight for corresponding motion, at t_n.

This incremental approach overcomes the possible errors emerged when there is a high jump in input parameters. There occurs a visible error in the blended motion, since a high jump in input parameter vector rapidly reduces the weights of motions that were high according to previous parameters and vice versa. Because of this rapid change, the position of the generated motion skips to the position of

the others. Although this method eliminates visible jumps in positions, jumps in orientations are still a problem if this jump is too high. This can be corrected by spreading the jump to upcoming frames by using a sinusoidal function. However, this reduces the flexibility of transitioning system. In other words, this approach would ignore the high jump request of the user, by replacing it with a sequence of smaller jumps. Although this problem is not in the scope of this study, we suggest applying anticipation algorithms, as described in [45].

4.2.3.2 Incremental Orientation Blending

We preferred representing orientations with quaternions, since using interpolation methods on Euler Angles may result into poor outputs. The main reason for that is the representation of a rotation is not unique when Euler Angles are used. Although the quaternions also have two representations, say Q_1 and Q_2 for an orientation, there is a relation between these values, that is $Q_1 = -Q_2$. This kind of ambiguity can be cleared, by selecting the representation that is closest to the corresponding quaternions of the other poses. The details for handling this ambiguity will be explained later.

In many approaches, Spherical Linear Interpolation is used for interpolating quaternions; however, it can only handle blends of size two. So, we employ the blending scheme for multiple motions, as described in [24]. The basic idea is to transform the orientation into a vector space $\Re^3$, with respect to a reference orientation. Then, a linear weighted interpolation is applied on positions (see §4.2.3.1), and finally the output vector is reverted back into orientation space.

In order to carry out the transform, we used logarithm and exponential maps (cf. Equations 2.3 and 2.4). A quaternion q is mapped into its corresponding displacement vector v with respect to a reference quaternion q_*, by using the logarithm map:

$$v = \log(q_*^{-1} q). \tag{4.19}$$

And it can be transformed back from displacement vector v as:

$$q = q_* \exp(v). \tag{4.20}$$

As mentioned earlier, there is a possibility that q may be on the opposite hemispheres on the sphere with q_*. In that case, we use the other representation of the same orientation, that is $-q$. However, working with more than one motion requires the reference quaternion q should be on the same hemisphere with all the quaternions $q_1, q_2, \ldots q_{N_e}$, where q_i is the corresponding orientation of motion i in the example set. To minimize the total distance of q_* to all other quaternions, the following distance metric is used:

$$dist(q_1, q_2) = \sin(\left\|\log(q_1^{-1} q_2)\right\|). \tag{4.21}$$

This distance metric is preferred since it is differentiable at every point between $[0, \pi]$. Using this metric, the sum of square distances at Equation 4.22 should be minimized to obtain q_*.

$$E = \sum_{i=1}^{N_e} \left\| dist(q_*, q_i)) \right\|^2, \tag{4.22}$$

where N_e is the number of example motions. This equation can be written as:

$$E = \sum_{i=1}^{N_e} \sin^2 \theta_i, \tag{4.23}$$

where $\theta_i = \|\log(q_*^{-1} q_2)\|$. Since $\sin^2(\theta_i) = (1 - \cos^2(\theta_i)$ and the $\cos(\theta)$ is the dot product of the quaternions, i.e. $\cos(\theta_i) = q_i^T \cdot q_*$, total error E can be written as:

$$E = \sum_{i=1}^{N_e} \left(1 - (q_i^T \cdot q_*)^2\right). \tag{4.24}$$

The Lagrangian multiplier method, with multiplier λ is employed to find q_* that minimizes E:

$$\frac{\partial E}{\partial q_*} = \lambda \frac{\partial \left(1 - \|q_*|^2\right)}{\partial q_*}. \tag{4.25}$$

Combining Equations 4.25 and 4.24, we have:

$$\left(\sum_{i=1}^{N_e} q_i \cdot q_i^T\right) q_* = \lambda q_* \quad or \quad Aq_* = \lambda q_*, \tag{4.26}$$

where A is a 4×4 matrix, and q_* is a 4×1 vector. By doing so the problem of finding q_* is reduced into finding the eigenvector of A that minimizes E. In order to blend the quaternions of each pose we need to calculate the reference quaternion for every frame, which decreases the speed of our system. Therefore, assuming that the adjacent frames in a motion have similar orientations, we calculate the reference quaternion only for first frame. For the rest of the frames, the output orientation of the previous frame is used, as shown in Equation 4.27:

$$q_*(t_n) = \begin{cases} q_{0*}, & if\ n = 1 \\ q_(t_{n-1}), & else \end{cases} \tag{4.27}$$

where q_{0*} is the reference quaternion for first frame, calculated as specified earlier. Given $q(t_{n-1})$ at frame $n-1$, the displacement vector $v_i(t_n)$ for $q_i(t_n)$ of ith motion, where $1 \leq i \leq N_e$ and N_e number of example motion, is formulated as follows:

$$v_i(t_n) = \log(q_*(t_n)^{-1} q_i(t_n)). \tag{4.28}$$

We determine the displacement vector $v(t_n)$ for the generated motion by blending the displacement vectors of all motions as:

$$v(t_n) = \sum_{i=1}^{N_e} w_i v_i(t_n), \tag{4.29}$$

where w_i is the weight for corresponding motion, at t_n. By applying the inverse transformation at Equation 4.20, we find the blended orientation $q(t_n)$ as follows:

$$q(t_n) = q_*(t_n) \exp(v(t_n)). \tag{4.30}$$

It should be noted that the blended orientation $q(t_n)$ will be used as the reference orientation for frame $n + 1$.

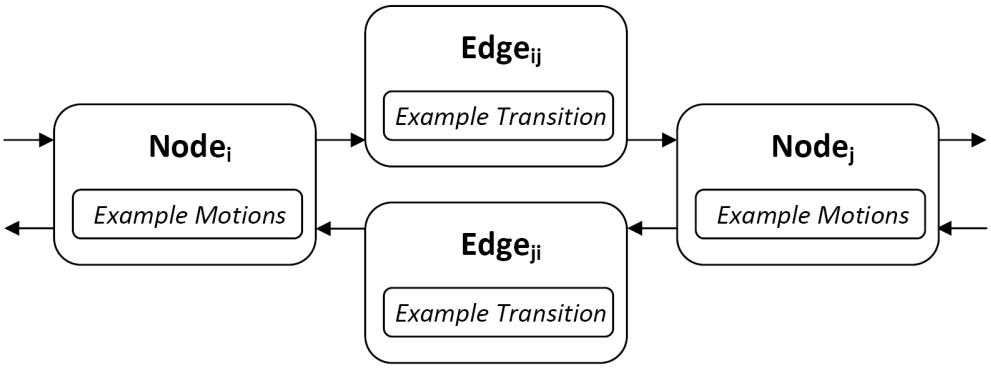


Figure 4.9: The illustration of the example motions in each node and the example transition on each edge.

4.2.4 Transition Handling

As the motion segments for each node are formed according to given parameters, the transition motions are created. Unlike other motions, transition motions are represented as edges, which ensures the connectivity of the graph (see Figure 4.9). The graph walk is converted into locomotion by concatenating the motions on the edges and the generated motions of the nodes on the graph one after another.

Transition motions are very similar to motions in the node groups. They also have keyframes, each of which contain a posture with ground contact. However, unlike other motions, not all of their postures with foot contact are selected as keyframes. Transition motions, basically consists of three parts. Let M_i and M_j be the output motions, which are generated according to user-specified parameters, of the nodes N_i and N_j, respectively; and T_{ij} be the transition motion on the edge E_{ij}, which connects node N_i and node N_j. Then, the three parts of motion T_{ij} can be described as follows:

- Part B_{ij} is the motion segment at the beginning of transition motion that will be blended with the motion of node N_i. The ending frame of part B_{ij} is fixed.

- Part C_{ij} is the core of the transition movement, and it is not blended with any other motion. Unlike other parts this part has exactly two keyframes

at the beginning and at the end. The frames in between may have foot contacts, but they are not labeled as keyframes. Both the beginning and ending frames of part C_{ij} is fixed.

- Part E_{ij} is the motion segment at the end of transition motion that will be blended with the motion of node N_j. The beginning frame of part E_{ij} is fixed.

Given that E_{ij} connects the nodes N_i and N_j, for constructing the transition motion T_{ij}, first, the beginning and ending frames of the transition should be found. Let the frame sets $B_i = [b_s^i, b_e^i]$ and $B_{ij} = [b_s^{ij}, b_e^{ij}]$ be the segments of M_i and T_{ij}, which will be blended, respectively. These frames on the border are, mandatorily, keyframes. With b_e^{ij} fixed, b_e^i is the largest keyframe of M_i, with $f(b_e^{ij}) = f(b_e^i)$ where function $f(g)$ returns the foot contact type of the frame g. Then, b_s^{ij} is the first keyframe, that $f(b_s^{ij}) = f(b_s^i)$, after a transition is scheduled. It should be noted that f function of B_{ij} is the time shifted version of f of M_i.

Similarly, the frame sets $E_j = [e_s^j, e_e^j]$ and $E_{ij} = [e_s^{ij}, e_e^{ij}]$ be the segments of M_j and T_{ij}, which will be blended, respectively. Again, with e_s^{ij} fixed, e_s^j is the smallest keyframe on M_j, such that $f(e_s^{ij}) = f(e_s^j)$. This time, the blending is kept as large as possible; Therefore e_e^{ij} and e_e^j are the farthest keyframes, ensuring the keyframes at the respective intervals have the same foot contact sequences.

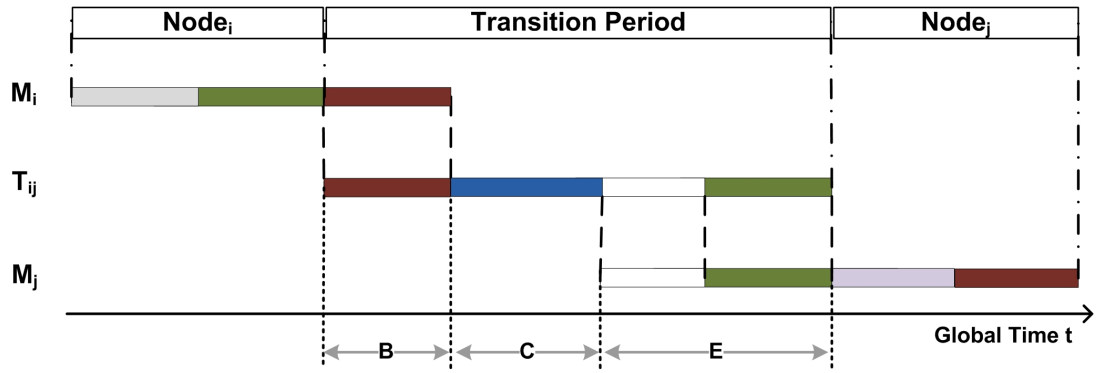


Figure 4.10: A transition period on the edge that connects Node$_i$ and Node$_j$.

As the segments of M_i, M_j, and T_{ij} to be blended is defined, we will define the blending scheme we apply on B and E parts of output motion, with B and E are

as shown in Figure 4.10. For this example, there are only two motions that should be blended; therefore, we do not need to use a data scattered interpolation based approach for blending. A linear interpolation based blending serves the purpose.

Before applying the blending scheme, the corresponding postures should be found. For this purpose, we use the synchronization approach described in §4.1.5. Moreover, since the position and the rotation of a motion depend the preceding motions on the walk, we need to transform the motion according to preceding motions, before blending it with the previous one. For this purpose, a global position, P_G and orientation, Q_G, is defined. Let M_k be the motion to be concatenated with foot position $p_k(t)$ and root orientation $q_k^0(t)$ represented as quaternion. M_k is first rotated by Q_G and moved by P_G before the interpolation is applied. It should be noted that start frame of M_k is positioned at the origin and its direction is aligned to x-axis. However, the transformation is applied according to frame i where the transition blending starts. Therefore, before re-orienting the motion we need to find the rotation,ΔQ_i, and position, ΔP_i of frame i according to frame 1:

$$\Delta Q_i = q_k^0(i)\ (q_k(1))^{-1}. \tag{4.31}$$

$$\Delta P_i = p_k(i) - p_k(1). \tag{4.32}$$

It should be noted that $\Delta P_i = p_k(i)$ since the motion is positioned at the origin at first frame. Then the rotation, $\tilde{q}_k^0(t)$ and position $\tilde{p}_k(t)$ of the posture at frame t after transformation can be calculated as follows.

$$\tilde{p}_k(t) = P_G + (Q_G)\ (\Delta Q_i)^{-1}\ (p_k(t) - \Delta P_i)\ (Q_i)\ (Q_G^*). \tag{4.33}$$

$$\tilde{q}_k^0(t) = (Q_G)\ (\Delta Q_i)^{-1}\ (q_k^0(t)). \tag{4.34}$$

Given P_i and P_{ij} as corresponding postures at B_i and B_{ij}, respectively, and $\tilde{P}$ as the corresponding posture of part B of the output motion $\tilde{T}_{ij}$. The blend $\tilde{p}$ of respective root positions, p_i and p_{ij}, is calculated using Linear Interpolation, that is:

$$\tilde{p} = \alpha p_i + (1 - \alpha) p_{ij}. \tag{4.35}$$

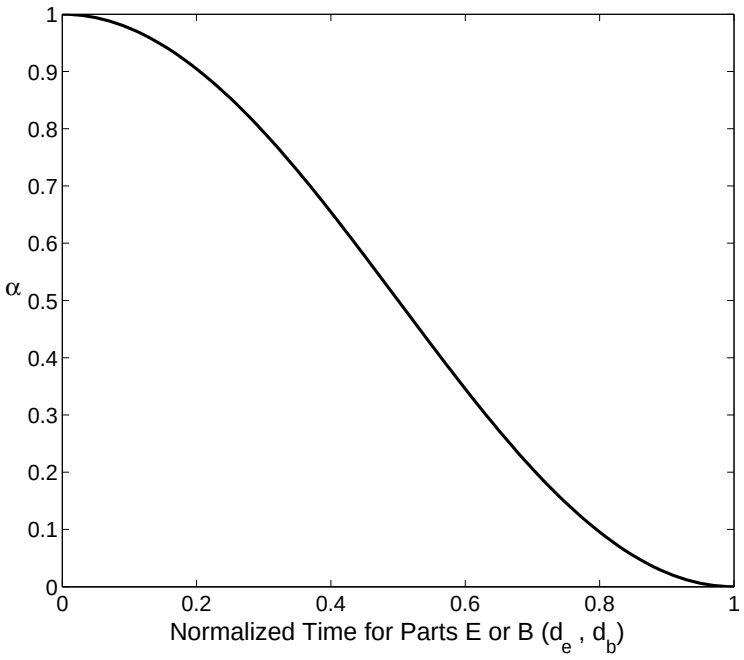


Figure 4.11: The α values for normalized time values of part B and part E: d_b and d_e, respectively, where $d_b = \frac{t-b_{start}}{b_{end}-b_{start}}$ or $d_e = \frac{t-e_{end}}{e_{end}-e_{start}}$.

For computing α, a sinusoidal function, as shown in Figure 4.11, is employed, that is,

$$\alpha = 0.5 + 0.5\cos\left(\pi\frac{t - b_{start}}{b_{end} - b_{start}}\right), \tag{4.36}$$

where b_{end} and b_{start} is the global end and start frames of output segment B, and t is the global frame number of postures P_i and P_{ij}. In order to blend the orientations, q_i and q_{ij}, Spherical Linear Interpolation is employed:

$$\tilde{q} = \text{Slerp}(q_i, q_{ij}, \alpha) \;=\; \frac{q_i \sin((1-\alpha)\theta) + q_{ij}\sin(\alpha\theta)}{\sin(\theta)}, \tag{4.37}$$

where $\theta = \arccos(q_i \cdot q_{ij})$, and is the half of the angle between q_i and q_{ij}.

Part E of the output motion $\tilde{T}_{ij}$ is formed very similarly. The position and orientation of each frame is computed using the same equations (Equation 4.35 and 4.37). Only difference is that, the formula for computing α (see Equation 4.36) is adapted in following way:

$$\alpha = 0.5 + 0.5\cos\left(\pi\frac{t - e_{start}}{e_{end} - e_{start}}\right), \tag{4.38}$$

where e_{end} and e_{start} is the global end and start frames of output segment E, and t is the global frame number of postures P_j and P_{ij}.

The core of the transition, part C, is imported into the output motion as it is, after the transformation.

4.2.5 Input Model

As the input, the user is asked to enter a sequence of motions and their respective parameters; e.g.,

```
Run  {2.0, [0: (3.0, 0)], [0.7: (3.5, 60)], [1.7: (3.0, 0)]},
Walk {1.0, [0: (2.7, 30)], [0.5: (1.1, 0)},
Stop,
...
```

The first part of each input segment defines the type of the motion, such as Run, Walk, Stop. The first number after the curly braces is the length of the motion on global time and it is followed by the parameter items, which are bounded with square brackets. A parameter item consists of parameters, in parentheses, and their timings. The transitions are handled automatically. Their duration should also be considered, while preparing an input sequence. Another issue is that the parameters are not applied on the exact timings given by the user. They are only applied if there is a foot contact, otherwise they are waited to be applied on the next foot contact. This approach should strictly be applied, in order to preserve the physically correctness of motions according to [28].

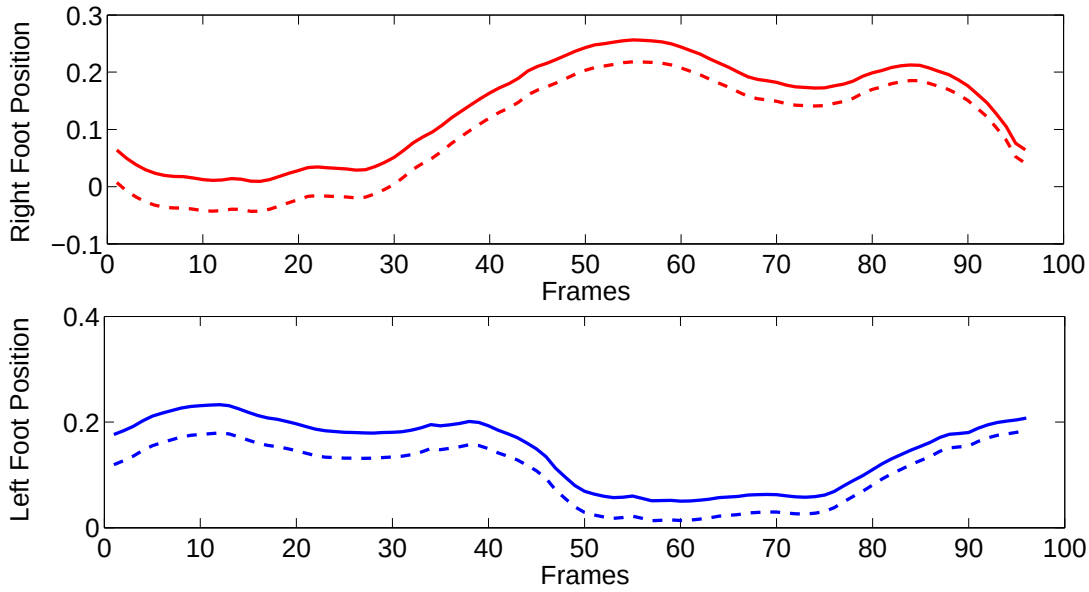
Chapter 5

Experimental Results

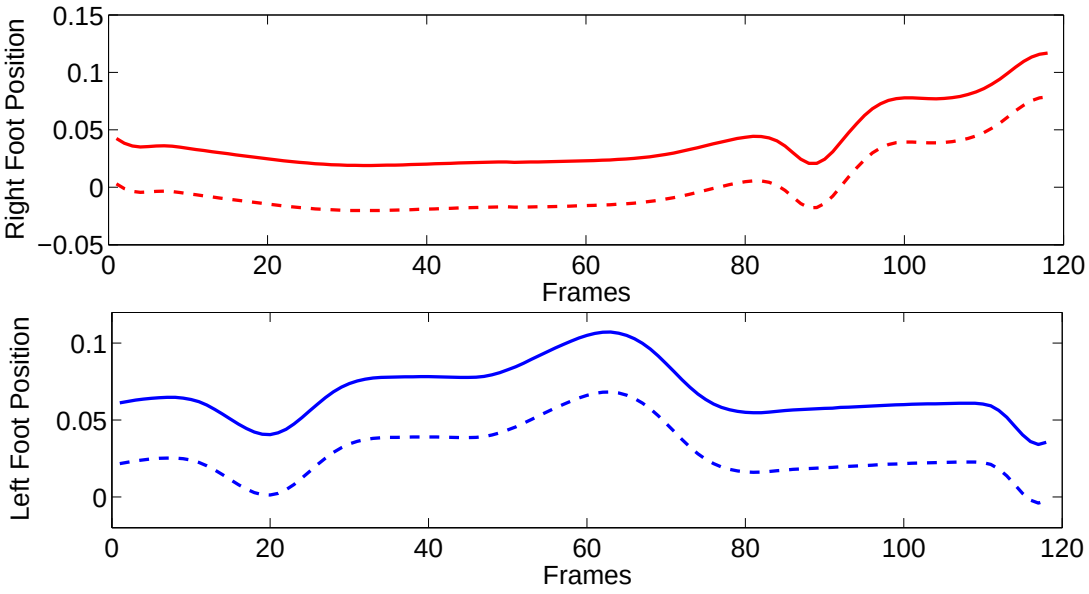
5.1 Results and Evaluation

In order to test our system, we performed experiments on various aspects of the system. First, we provide brief information about our implementation and test platform. Then, the results for tests on subjects, such as foot positioning procedure, weight calculation schemes, are provided. We also illustrate the resulting motions with sample still frames from animations. In our experiments, we used nineteen example motions: nine walking, nine running, and a standing motion. The walking and running motions differ from each other in linear and angular velocities. The skeletal model, (see Figure 4.8), consists of 62 DOFs: 6 DOF's for the root position and orientation, 3 DOFs for the head, 15 DOFs for the spine and neck, 24 DOFs for the arms, and 14 DOFs for the legs. The experiments are performed on a PC with Pentium IV 2.8 GHz CPU and 1.5 Gb memory.

In our experiments, the foot positioning approach gives feasible results most of the time. In Figure 5.1, the displacements for the toes are demonstrated. In general, the repositioned postures with foot contact are attached to ground successfully. However, in some motions, the captured data had little errors on z rotation of the posture and the minimum foot positions were unbalanced. We tried to rotate back the posture, but it resulted in a tilted motion. Nevertheless,



(a)



(b)

Figure 5.1: The positions of left and right toes of the motions are shown with solid line while the corrected positions are represented by dotted line: (a) the correction on a run motion, and (b) the correction on a transition motion.

these errors are relatively small and do not affect the output motion. In Figure 5.7, the effect of foot planting error correction method is shown on an example motion.

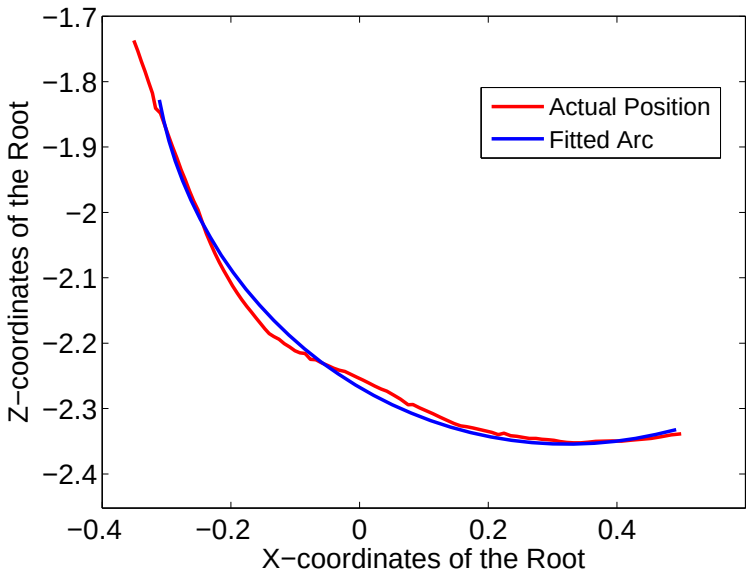


Figure 5.2: A successful arc fitting of root position trajectory.The red line shows the trajectory, while the blue line is the fitted circular arc.

We computed the linear and the angular velocity parameters, as specified in §4.1.3. The extracted parameters of example motions are shown in Table 5.1. In Figure 5.2, successful examples for arc fitting method for angular speed extraction are shown. In our tests, it is observed that the angle extracting approach fails if the circular motion does not have a stable angular velocity. Therefore, it requires a database that has MoCap data which is captured with the knowledge of latter usage. However, our database has arbitrary motions, and cleaning or pruning processes for the motions are strictly required. In Figure 5.3, the motion trajectories that have failed to be perceived as circular arcs are illustrated.

We also tested our weight calculation scheme. In Figure 5.4, the output weight values vs. normalized angular velocities (w_i) are shown, for linear velocity is fixed to 3.64. In Figure 5.5, the γ values for the same parameter values are depicted. It should be noted that in both of the figures the total weight value is always one, moreover the weights for the incremental time update (γ) are always non-negative which ensures the update is also non-negative. In Table 5.2, the computed weight

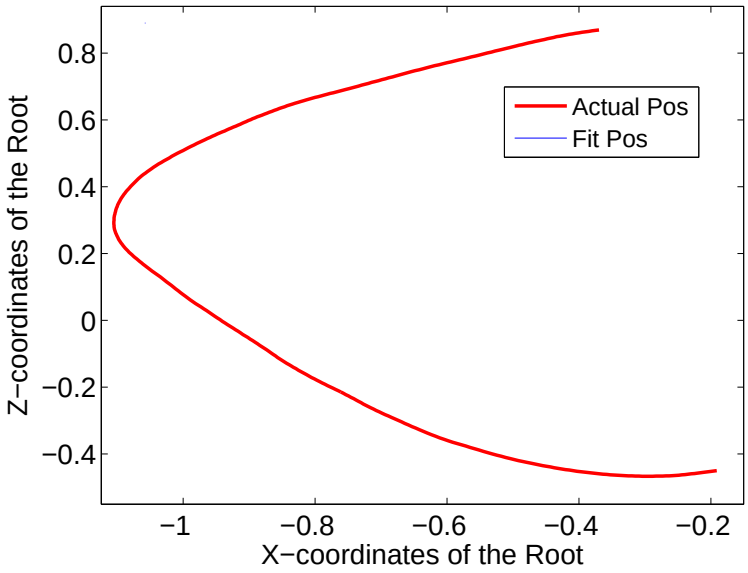


Figure 5.3: An unsuccessful arc fitting of root position trajectory. The red line shows the trajectory. Since the system cannot fit the trajectory to an arc, blue line for the fitted circular arc is not drawn.

		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9
Run	ω	4	5	6	55	-52	84	-67	42	-35
	v	2.8	3.7	5.2	4.3	4.2	3.3	3.6	2.8	2.9
Walk	ω	4	2	7	26	-22	74	-87	42	-35
	v	0.6	1.2	1.6	0.8	0.7	0.9	1.0	1.2	1.2

Table 5.1: Example motion parameters, where M_i is the i^{th} motion, ω is the angular velocity and v is the linear velocity.

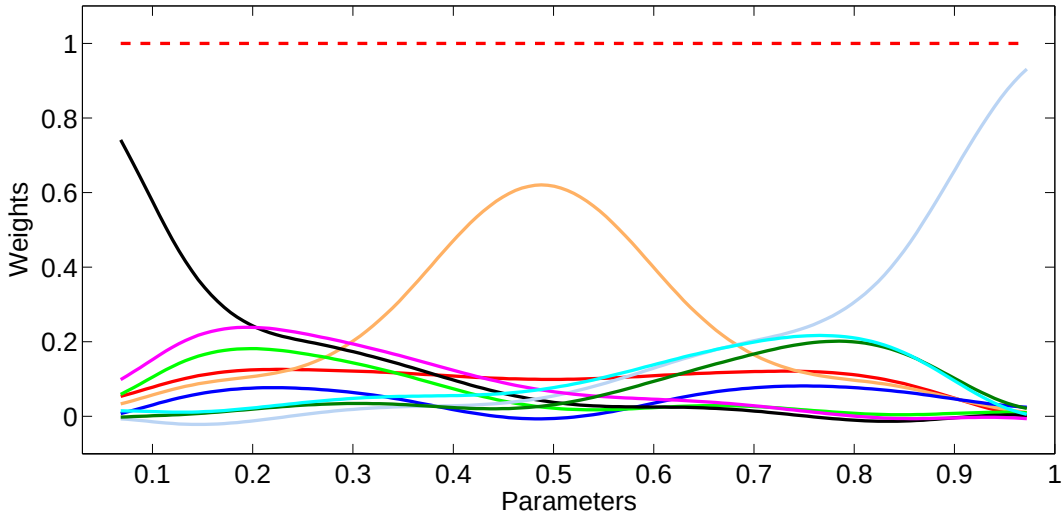


Figure 5.4: The weight values (w) vs. the normalized angular velocities. The solid lines represent the weights for example motions and the dotted line shows the overall sum.

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9
$\gamma(p)$	0.24	0.21	0	0.10	0.05	0.21	0.22	0.11	0.03
$w(p)$	0.19	0.25	0	-0.10	-0.05	0.21	0.12	0.11	0.03

Table 5.2: The posture blending weights (w) and the time update function weights (γ) of example motions M_i for the user-specified parameter p with $v = 3.4$ and $\omega = -30$.

values (w, γ) for an arbitrary parameter p.

We tested our system for several parameters and motion sequences. In most of the cases, we obtained motions that are capable of conforming to the given parameters. In Figures 5.8, . . ., 5.12, some of the generated motions are depicted. Due to the limitations on the original database example motion parameters are not well distributed; hence, in some cases extrapolation produces unrealistic results.

In Figure 5.6, the output parameters, which are calculated using linear interpolation with pre-computed weights, are compared with user-specified parameters. As shown in these figures, the weighting scheme produces the user-specified parameters by blending example parameters successfully.

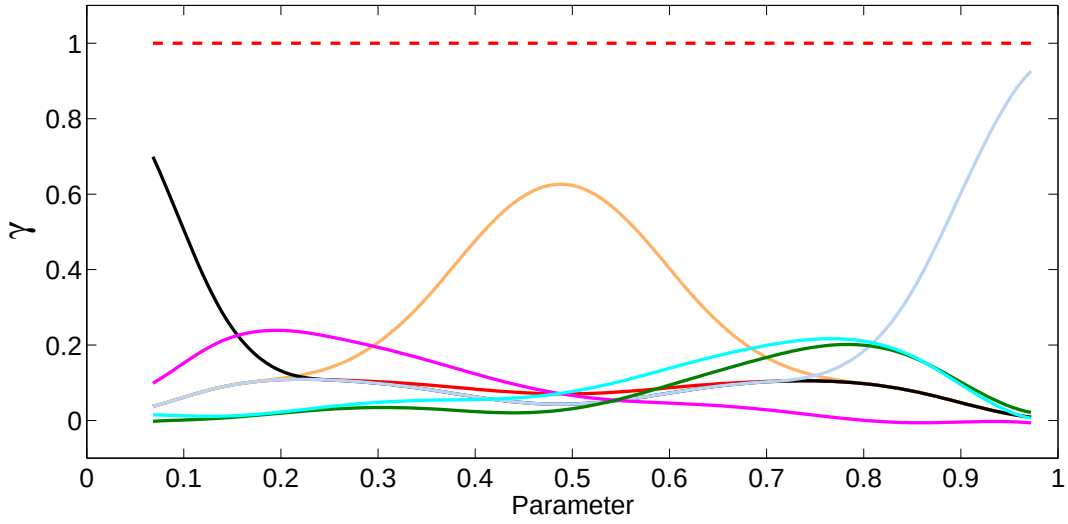
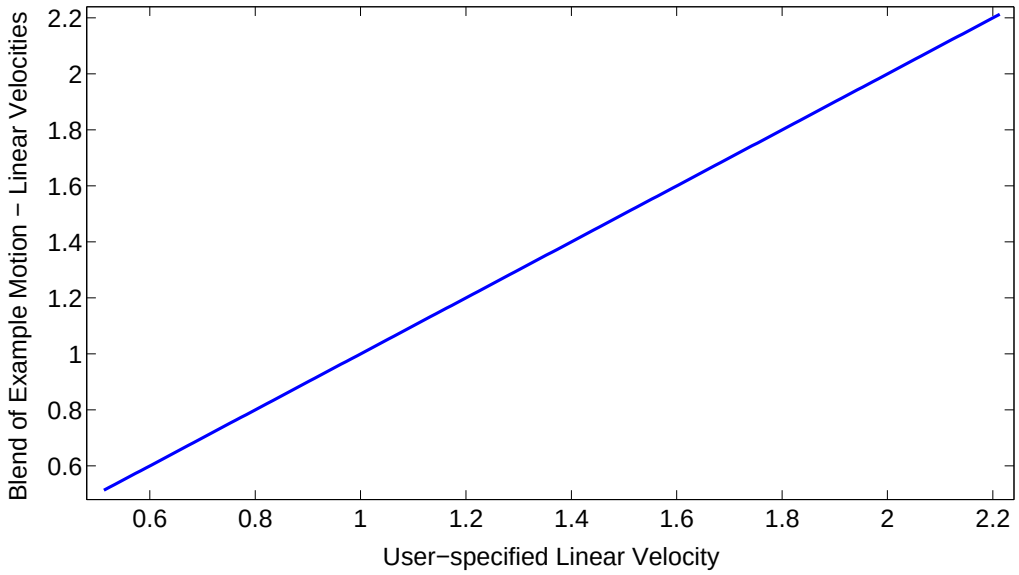


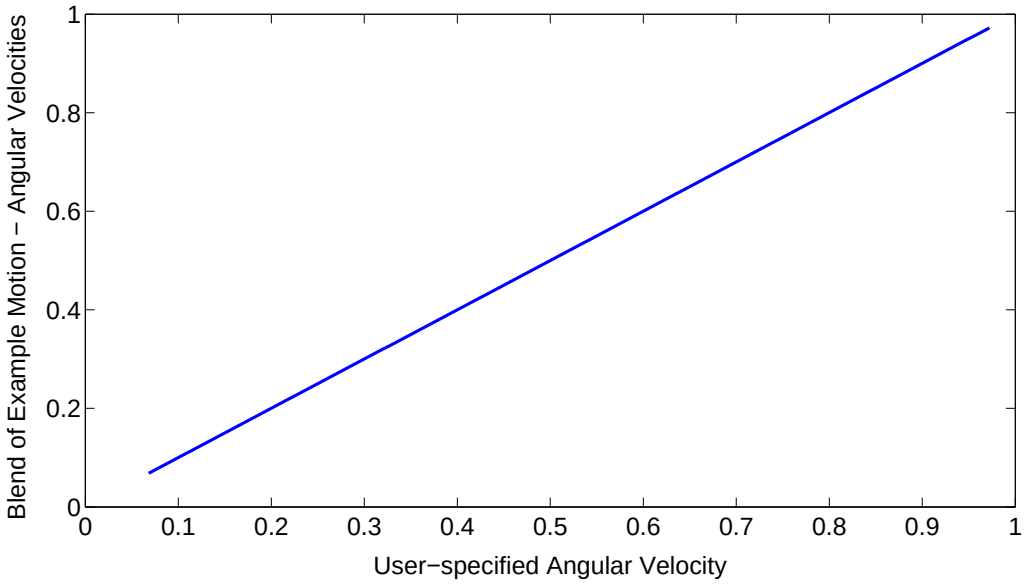
Figure 5.5: The weight values for incremental time update (γ) vs. normalized angular velocities. The solid lines represent the weights for example motions and the dotted line shows the overall sum.

Our incremental posture blending scheme produces smooth motions for quick changes in parameter sequences. As seen in Figure 5.9, while the normal approach fails to produce continuous motions, our method gives better results.

The longer motion clips are generated successfully using transition generation method, as illustrated in Figures 5.10, 5.11, and 5.12. In the majority of our experiments, smooth motion transitions are obtained. However, in some transitions, that have quick changes in speed, the results were not as good. In our opinion, this is the consequence of having single motions for transitions. As a future extension we plan on employing more than one example motion for each edge, and using the difference between the parameters of the preceding and the successor nodes as input parameter.



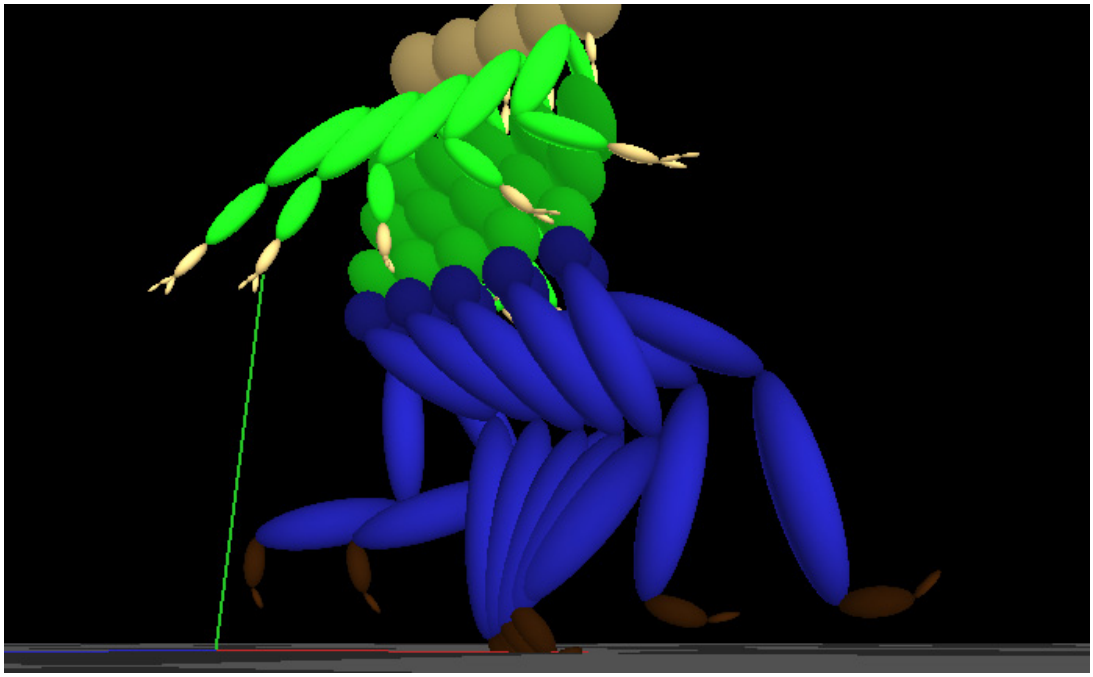
(a)



(b)

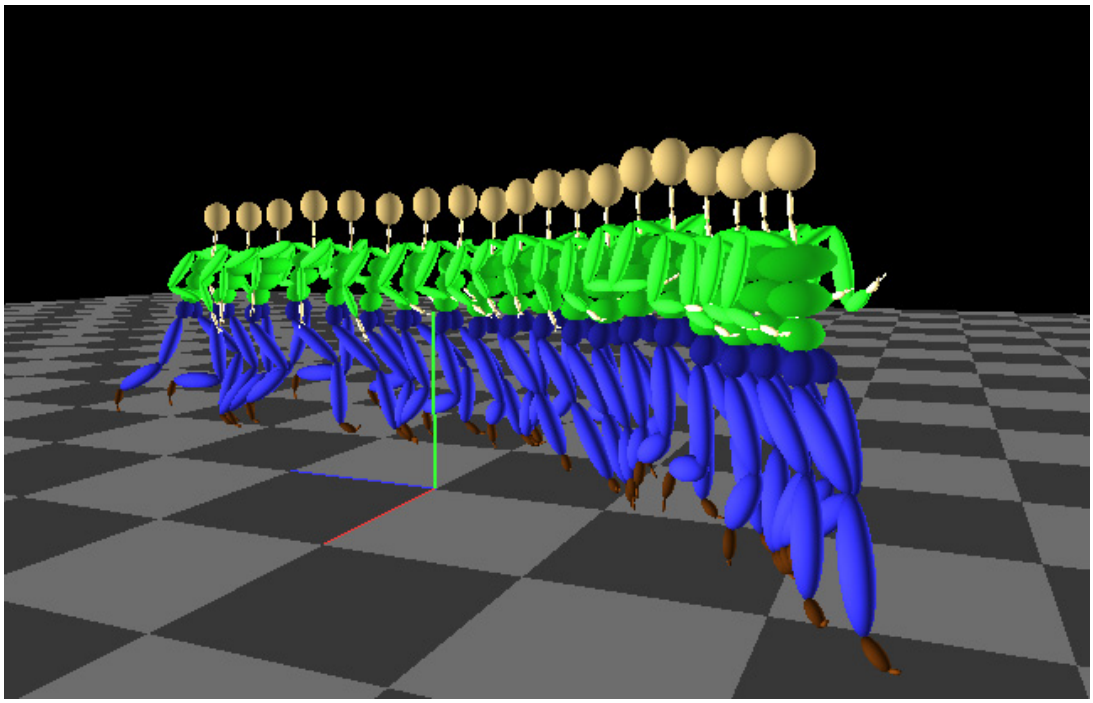
Figure 5.6: The user-specified parameters and the corresponding output parameters generated by weighted blending the parameters of the example motions: (a) the linear velocities are compared, with a fixed arbitrary angular velocity, (b) the angular velocities are compared, with a fixed arbitrary linear velocity.

(a)

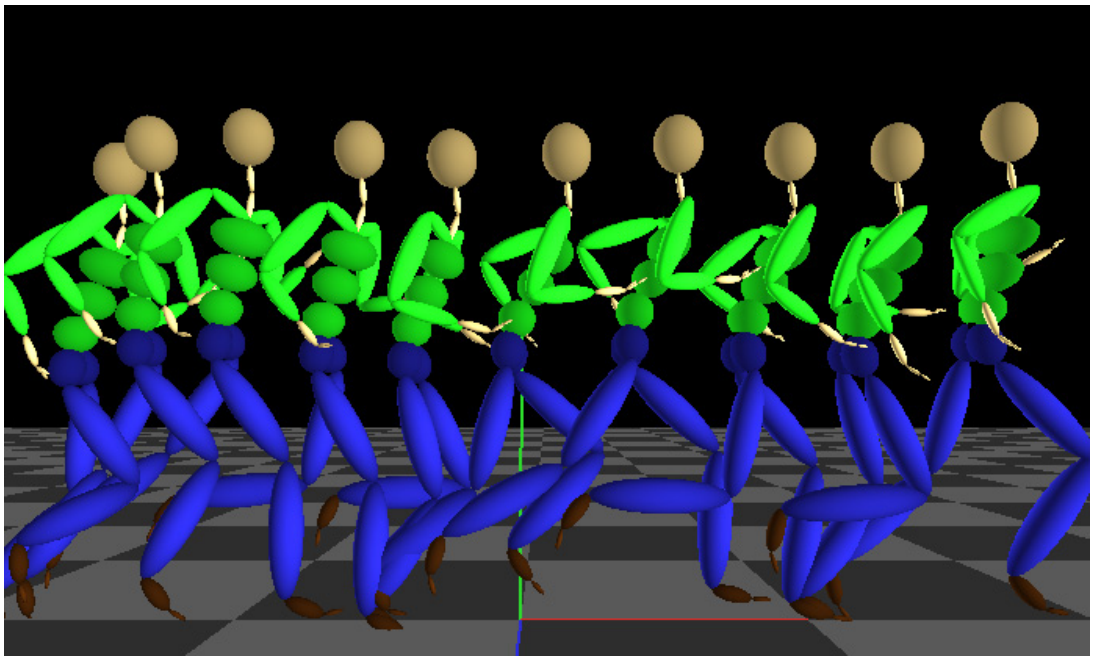


(b)

Figure 5.7: The foot position correction is shown on an example motion: (a) before correction, (b) after correction.

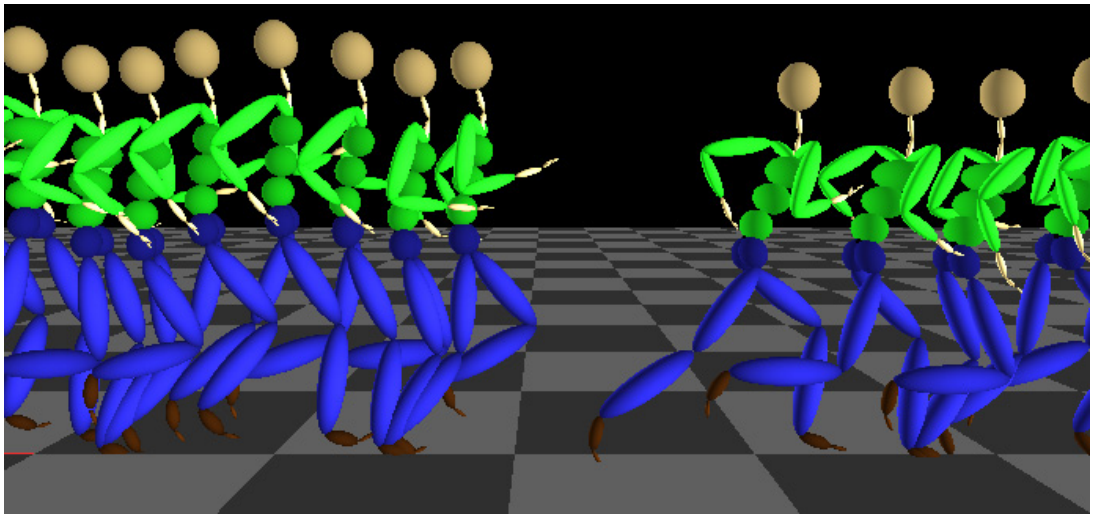


(a)

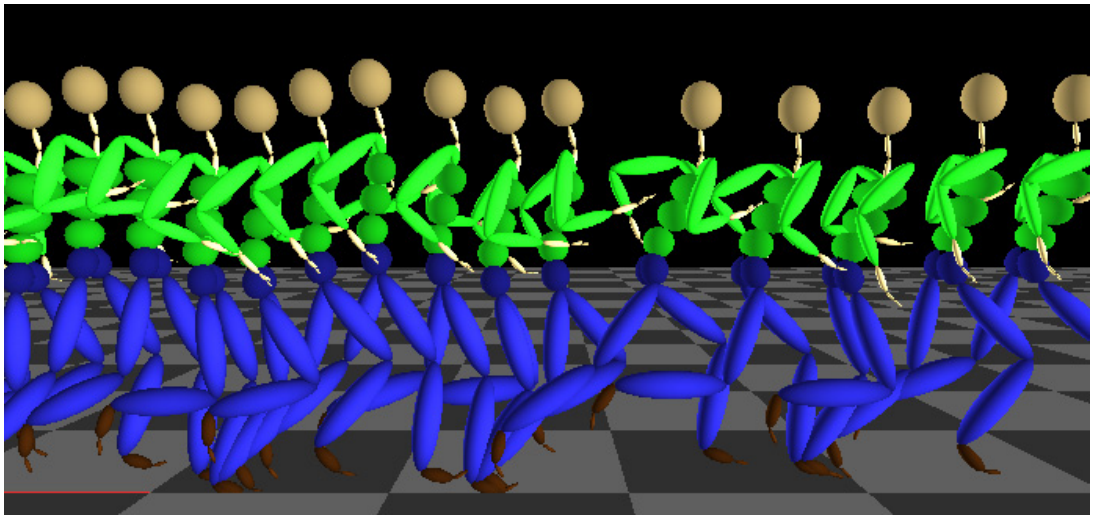


(b)

Figure 5.8: Motions with different angle and speed parameters: (a) running motion with changing speed, (b) running motion with changing speed and angle.



(a)



(b)

Figure 5.9: The illustration of incremental position blending on a running motion: (a) a motion generated with normal position blending, (b) a motion generated with incremental position blending with the same input parameters.

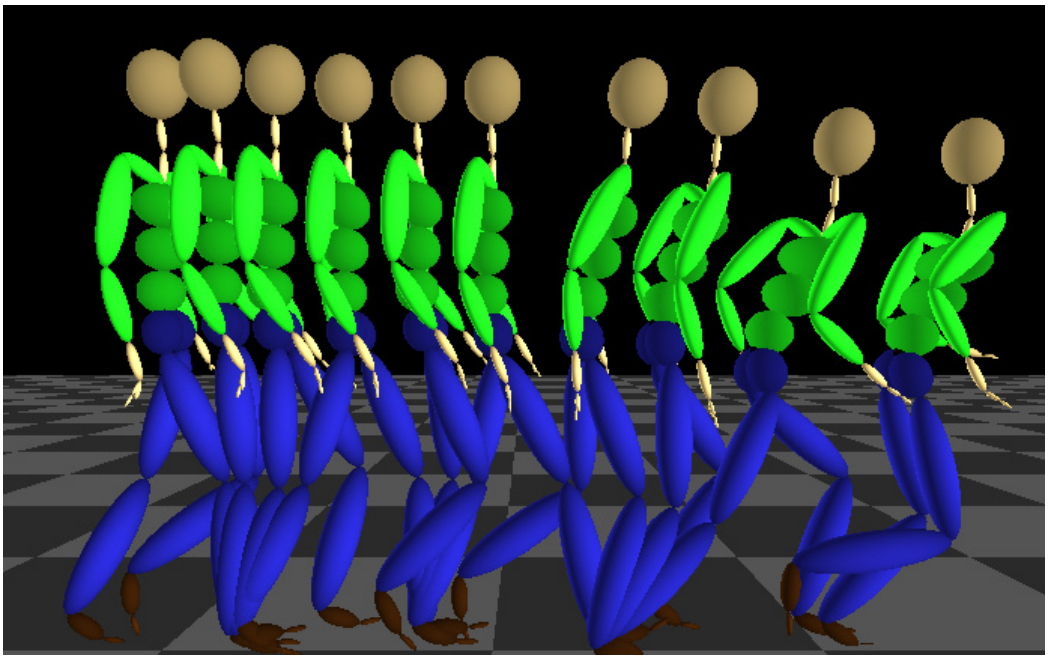


Figure 5.10: An example of walk-to-run transition motion, generated by our system.

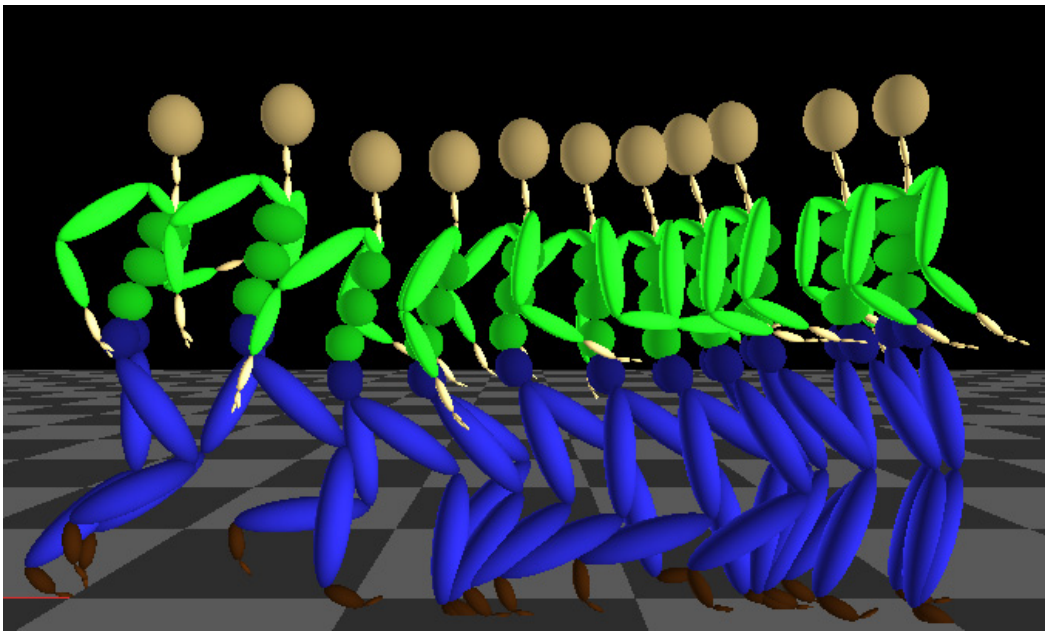


Figure 5.11: An example of run-to-stop transition motion, generated by our system.

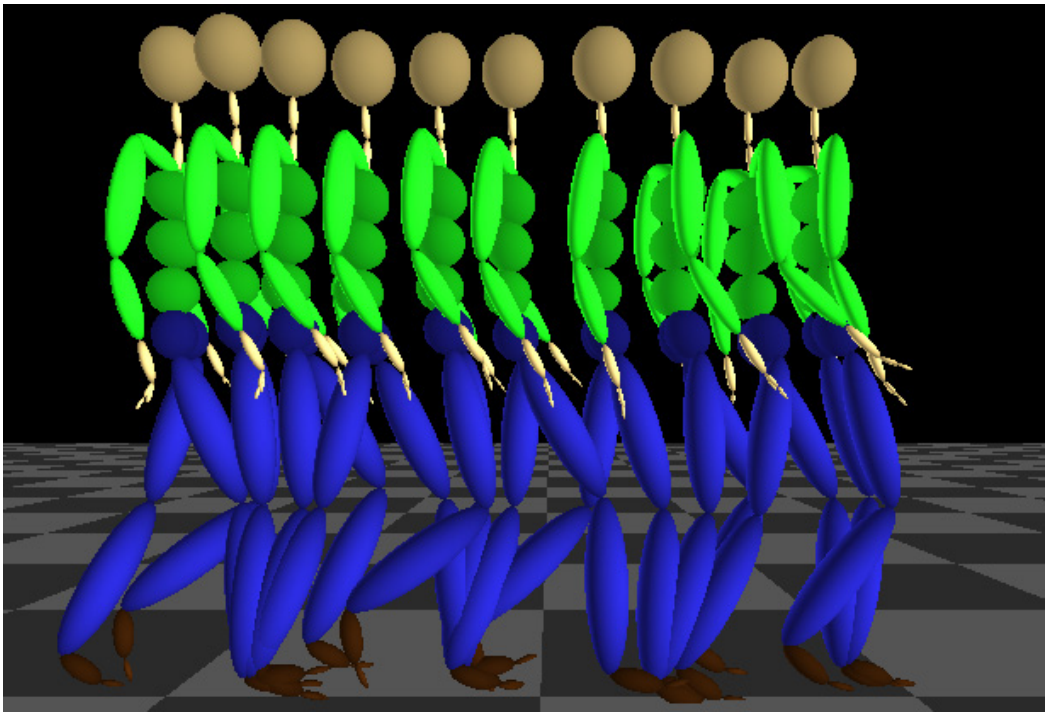


Figure 5.12: An example of walk-to-stop transition motion, generated by our system.

Chapter 6

Conclusion

Locomotion is one of the most frequently used actions in daily life. Therefore, modeling of motions of this kind is very important in animating humans. In this thesis, we proposed a system that creates locomotion according to the user-specified parameters. The system works on real-time. Motion graphs provides an effective way for utilizing large motion sets, hence, we registered the motions in our dataset to a motion graph. For constructing the motion graph, several techniques that serve the system in different aspects are employed. Moreover, a method is proposed for using these techniques systematically. With this method, it is easy to utilize the datasets that have low quality motions, which are poorly categorized and not parameterized.

The proposed system consists of two stages: on-line and off-line stages. This kind of separation increases the run-time efficiency of the system by reducing the respective processes. Also, it is important for the comprehensibility of the system. At off-line stage the motion graph is constructed, while at on-line stage an output motion is generated according to user-specified parameters using the motion graph. Unlike other approaches, the error correction is handled in off-line stage. This increases the speed of on-line step without disturbing the expected motion quality. At on-line stage, we used incremental posture blending. This approach allows rapid changes in parameter space below a threshold.

In future studies, the interface for the system can be improved. Medium level controls such as following mouse can be implemented over the current system, as well as some high level controls such as path planning. The system is also expandable in parameter space. Therefore, some other parameters such as style or terrain information can also be added.

Although the current implementation may not satisfy a professional animator in industry due to the limited motion variety and its low level interface, we believe that it provides basic tools and interface for researches and amateur studies. Moreover, since it works with keyframes and is easy-to-expand, it can be easily incorporated into other animation platforms as a locomotion generation tool.

Bibliography

[1] Y. Abe, C. K. Liu, and Z. Popović. Momentum-based parameterization of dynamic character motion. *Graphical Models (Special Issue on ACM/Eurographics Symposium on Computer Animation 2004)*, 68(2):194–211, 2006.

[2] B. Allen, B. Curless, Z. Popović, and A. Hertzmann. Learning a correlated model of identity and pose-dependent body shape variation for real-time synthesis. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 147–156, 2006.

[3] O. Arikan, D. A. Forsyth, and J. F. O'Brien. Pushing people around. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 59–66, 2005.

[4] N. I. Badler, C. B. Phillips, and B. L. Webber. *Simulating Humans: Computer Graphics, Animation and Control*. Oxford University Press, Inc., 1993.

[5] B. Bodenheimer, C. Rose, S. Rosental, and J. Pella. The process of motion capture: Dealing with the data. In D. Thalmann and M. van de Panne, editors, *Proceedings of the Eurographics Workshop on Computer Animation and Simulation 97*, pages 3–18, September 1997.

[6] D. C. Brogan, R. A. Metoyer, and J. K. Hodgins. Dynamically simulated characters in virtual environments. *IEEE Computer Graphics and Applications*, 18(5):58–69, 1998.

[7] A. Bruderlin and L. Williams. Motion signal processing. In *ACM Computer Graphics (Proceedings of SIGGRAPH '95)*, pages 97–104, 1995.

[8] N. Burtnyk and M. Wein. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. *Communications of the ACM*, 19(10):564–569, 1976.

[9] K.-J. Choi and H.-S. Ko. On-line motion retargetting. In *Proceedings of the Pacific Graphics '99*, pages 32–42, 1999.

[10] M. da Silva, Y. Abe, and J. Popović. Interactive simulation of stylized human locomotion. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '08)*, 27(3), Article No. 82, 2008.

[11] A. C. Fang and N. S. Pollard. Efficient synthesis of physically valid human motion. *ACM Transactions on Graphics*, 22(3):417–426, 2003.

[12] M. Girard and A. A. Maciejewski. Computational modeling for the computer animation of legged figures. *ACM Computer Graphics (Proceedings of SIGGRAPH '85)*, 19(3):263–270, 1985.

[13] F. S. Grassia. *Believable Automatically Synthesized Motion by Knowledge-enhanced Motion Transformation*. PhD thesis, School of Computer Science, Carnegie Mellon University, 2000.

[14] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović. Style-based inverse kinematics. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '04)*, 23(3):522–531, 2004.

[15] J. K. Hodgins and N. S. Pollard. Adapting simulated behaviors for new characters. In *ACM Computer Graphics (Proceedings of SIGGRAPH '97)*, pages 153–162, 1997.

[16] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien. Animating human athletics. In *ACM Computer Graphics (Proceedings of SIGGRAPH '95)*, pages 71–78, 1995.

[17] L. Kovar and M. Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Transactions on Graphics (Proceedings of SIGGRAPGH '04)*, 23(3):559–568, 2004.

[18] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '02)*, 21(3):473–482, 2002.

[19] T. Kwon and S. Y. Shin. Motion modeling for on-line locomotion synthesis. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 29–38, 2005.

[20] J. Lasseter. Principles of traditional animation applied to 3d computer animation. In *ACM Computer Graphics (Proceedings of SIGGRAPH '87)*, pages 35–44, 1987.

[21] J. Lee and S. Y. Shin. A hierarchical approach to interactive motion editing for human-like figures. In *ACM Computer Graphics (Proceedings of SIGGRAPH '99)*, pages 39–48, 1999.

[22] C. K. Liu and Z. Popović. Synthesis of complex dynamic character motion from simple animations. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '02)*, 21(3):408–416, 2002.

[23] M. Neff and E. Fiume. Modeling tension and relaxation for computer animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 81–88, 2002.

[24] S. I. Park, H. J. Shin, and S. Y. Shin. On-line locomotion generation based on motion blending. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 105–111, 2002.

[25] K. Perlin. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):5–15, 1995.

[26] Z. Popović and A. Witkin. Physically based motion transformation. In *ACM Computer Graphics (Proceedings of SIGGRAPH '99)*, pages 11–20, 1999.

[27] C. Rose, M. F. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–40, 1998.

[28] A. Safonova and J. K. Hodgins. Analyzing the physical correctness of interpolated human motion. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 171–180, 2005.

[29] A. Safonova and J. K. Hodgins. Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '07)*, 26(3), Article No. 106, 2007.

[30] A. Safonova, J. K. Hodgins, and N. S. Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '04)*, 23(3):514–521, 2004.

[31] H. J. Shin, J. Lee, S. Y. Shin, and M. Gleicher. Computer puppetry: An importance-based approach. *ACM Transactions on Graphics*, 20(2):67–94, 2001.

[32] K. Shoemake. Animating rotation with quaternion curves. *ACM Computer Graphics (Proceedings of SIGGRAPH '85)*, 19(3):245–254, 2006.

[33] P.-P. J. Sloan, I. Charles F. Rose, and M. F. Cohen. Shape by example. In *Proceedings of the Symposium on Interactive 3D Graphics (I3D '01)*, pages 135–143, 2001.

[34] D. Tolani, A. Goswami, and N. I. Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models*, 62(5):353–388, 2000.

[35] M. Unuma, K. Anjyo, and R. Takeuchi. Fourier principles for emotion-based human figure animation. In *ACM Computer Graphics (Proceedings of SIGGRAPH '95)*, pages 91–96, 1995.

[36] J. Wang, S. M. Drucker, M. Agrawala, and M. F. Cohen. The cartoon animation filter. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '06)*, 25(3):1169–1173, 2006.

[37] L. Wang and C. Chen. A combined optimization method for solving the inverse kinematics problem of mechanical manipulators. *IEEE Transactions On Robotics and Applications*, 7(4):489–499, 1991.

[38] C. Welman. Inverse Kinematics and Geometric Constraints for Articulated Figure Manipulation. Master's thesis, Simon Fraser University, 1993.

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

[40] D. A. Winter. *Biomechanics and Motor Control of Human Movement*. Wiley, August 2004.

[41] A. Witkin and M. Kass. Spacetime constraints. In *ACM Computer Graphics (Proceedings of SIGGRAPH'88)*, pages 159–168, 1988.

[42] W. L. Wooten. *Simulation of Leaping, Tumbling, Landing, and Balancing Humans*. PhD thesis, Georgia Institute of Technology, 1998.

[43] W. L. Wooten and J. K. Hodgins. Simulation of human diving. In *Proceedings of Graphics Interface '95*, pages 1–9, 1995.

[44] X. Zhao. *Kinematic Control of Human Postures for Task Simulation*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1996.

[45] V. Zordan, A. Macchietto, J. Medin, M. Soriano, C.-C. Wu, R. Metoyer, and R. Rose. Anticipation from example. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST '07)*, pages 81–84, 2007.

[46] V. B. Zordan, A. Majkowska, B. Chiu, and M. Fast. Dynamic response for motion capture animation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH '05)*, 24(3):697–701, 2005.