Technical Report

UCAM-CL-TR-XXX
ISSN XXX-XXX

Number XXX

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Automated Motion Editing

Marc Cardle
King's College

May 2004

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500
http://www.cl.cam.ac.uk/

© 2004 Marc Cardle

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/TechReports/

ISSN XXXX-XXXX

ABSTRACT

Producing content for film, TV, animation, virtual environments or video games is a labour-intensive process. The primary goal of this research is to simplify and automate the production and re-use of animation.

In the context of animation production, the popularity of human motion capture in computer movies and three-dimensional computer games has motivated the development of many techniques to tackle the difficult and laborious problem of motion editing. The existence of large libraries of captured human motion has resulted in a growing demand for content-based retrieval of motion sequences without using annotations or other meta-data. This work addresses the issue of rapidly retrieving perceptually similar occurrences of a particular motion in a long motion sequence or unstructured motion database for the purpose of replicating editing operations with minimal user input. Editing operations on a single motion are made to affect all similar matching motions.

These techniques demonstrate that complex and repetitive operations can be automated over motion domain with little user intervention. The algorithms draw on signal processing, time-series matching, motion editing and statistical learning.

TABLE OF CONTENTS

CHAPTER 4 RESULTS

CHAPTER 5 CONCLUSION AND FUTURE WORK

APPENDICES

Chapter 1

INTRODUCTION

1.1 Background

There has been a digital revolution in films and video games during the last 20 years. This revolution is the shift from analogue to digital methods of recording and manipulating visuals and sound. The following section describes in more detail the advent of human motion capture and its uses in production.

1.1.1 Motion Capture

Computers have become an important part of the film-maker's toolkit. The computer's use as an audiovisual production and manipulation tool has changed the industry. Scenes that were previously impossible are now produced with such realism that the audience is unaware of any digital intervention. Superhuman stunts, fantastic special effects and the bringing to life of 3D animated characters are all made possible by computers. But computer generated imagery (CGI) is not limited anymore to the secondary role of creating realistic effects for films such as Sony Digital Imageworks' *Spiderman*. It is becoming a medium in itself and is also creating a whole new market for exclusively CGI animated features. While 1995 was the year that marked the release of *Toy Story*, the first all CGI animated feature, 2004 is poised to be the most productive year yet for CGI animated features: *Shrek 2* from PDI/Dreamworks (Figure 1) and *The Incredibles* from Pixar. In 2003, *Finding Nemo* grossed $340 million making it the ninth best selling films of all time [Boxofficemojo.com 2003]. Understandably, Pixar and PDI/Dreamworks are now currently exclusively focused on 3D animation. Dreamworks' traditional 2D animation studio in Los Angeles is in the process of switching over to 3D work. Even Walt Disney, a bastion of hand-drawn 2D cell animation, closed its Orlando studio in 2003 that had previously worked on popular hand-animated films such as *Lilo & Stitch* and *Mulan*.

However, CGI continues to be difficult to use. Modern CGI owes some of its realism and throughput to the recent developments in motion capture technology. Motion capture [Sturman 1994] is the recording of motion for immediate or delayed analysis and playback. Example motions are made by capturing high-fidelity recordings of the 3D movement from a live performer. Motion-capture is able to create realistic animation, which would be difficult to achieve using key-frame techniques [Baumgartner 1999]. Human motion is so complex that model-based animation techniques still do not produce realistic motions.

Figure 1: *DreamWorks' Shrek 2*

Motion capture (or *mocap*) is seen as a time and cost effective alternative to traditional keyframing [Robertson 1999] where the average animator will produce only two to five seconds of keyframed animation per character per day. On the other hand, the animation studio Protozoa obtained 15 minutes of motion capture for each of the 13 episodes of *And for The Dog and Dinosaur Show* in two weeks only [Morales 2001]. Despite the evident benefits, motion capture is not completely replacing keyframe and 3D animation, since these traditional approaches give more expressive control to the animator. Leading animators worldwide are reaching the consensus that a combination of techniques often works best [Panels Session 2002].

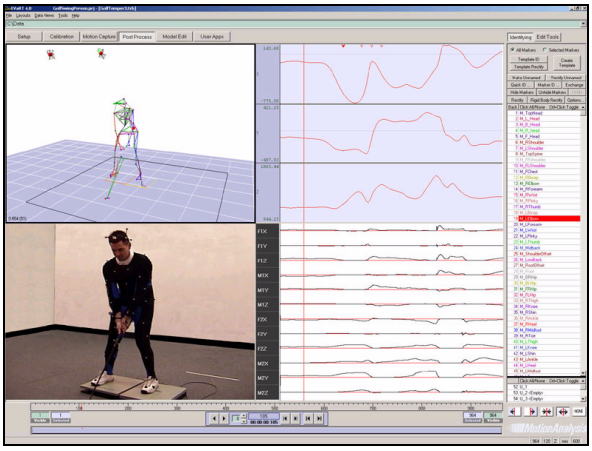
Figure 2: *Motion Analysis*' Capturing Software.

Motion capture is driven by live-action performers dressed with an array of sensors or markers that allow every motion to be tracked and recorded (Figure 2). The movements appear as dots on a computer screen and software is later used to locate the moving dots forming the skeleton of the person's body. From there, animators augment the skeleton with muscle tone and skin to give the virtual character a lifelike feel. Several technologies are currently available to capture motion including optical, magnetic, inertial, and laser, although optical and magnetic dominate the motion capture animation field. A detailed review of these is

presented by Menache [2000]. These systems remain expensive ranging from $30,000 for a magnetic unit to $200,000 for fully outfitted optical tracking system. However, the quality is increasing and costs are falling since motion capture is now commonplace in non-animated blockbuster films. For example, it was used for massive crowd scenes of virtual warriors such as in *Lord of the Rings* (Figure 3 (right)). It is also used to replace real actors in costly and dangerous stunt work, such as when Digital Domain portrayed victims falling to their death on the doomed *Titanic*, or to create completely virtual characters as in *The Matrix Reloaded* (Figure 3 (left)).

Figure 3: Motion captured virtual actors in *The Matrix Reloaded* (*left*) and in *The Lord of the Rings – Return of the King* (*right*).

Television uses motion capture for real-time character animation transmitted live to audiences. The advent of motion capture has also changed the way games are produced. Programmers used to have to draw every single keyframe of animation painstakingly, and are now able to produce similar sequences rapidly with motion capture. In games, the rendered graphics need to react to the player's movement; motion capture makes it possible to generate realistic animation. Electronic Arts was the first to introduce motion capture in video games in order to enhance the animations of their football players in *John Madden Football 1995* (Figure 4(left)). Recently, the producers Pinkett and Smith from *The Matrix Reloaded* devoted extra time to filming some of the 4,000 motion-capture scenes for the *Enter the Matrix* video-game (Figure 4(right)).

Figure 4: *Electronic Art*'s motion captured players in *John Madden 2003 NFL Football*. Motion captured monsters in *Enter the Matrix*.

Lower costs and improved toolsets are making motion capture more widely available. Applications for motion capture are consequently rapidly growing into a variety of other fields (Figure 5):

Medicine. For orthopaedists and prosthetists, it is useful to obtain quantitative gait analysis and an objective documentation of walking ability.

Sport science. For improving athletes' techniques, such as in golf or running.

Military. To train military personnel.

Architecture. For virtual walkthroughs containing virtual crowds.

Virtual Reality. For live tele-presence and realistic avatars.

Digital arts. For dancers to drive graphics or sound generation.

Education. For photo-realistic 3D virtual teachers and students.

Figure 5: New application areas of motion capture.

Accessibility and technological improvements have made large mocap datasets increasingly commonplace. The ability to build reusable motion libraries from these datasets is also a reason for using motion capture: once the motion is captured it can be used in many different contexts [Morales 2001]. This maximizes the gains obtained from the time and financial investment of capturing sessions. Consequently, a significant amount of effort has focused on improving the reusability of motion capture. This include motion retargeting, high-level editing and novel motion synthesis techniques. However, many of these techniques are ill-adapted to deal with large databases.

Asset management software from Kinetic Impulse and NXN Alienbrain were introduced to help manage the sheer amount of mocap data for easier manipulation, storage and retrieval. As in the recent research by Grünvogel et al. [2002], these systems assume that each mocap sequence in the database is annotated with respect to a classification scheme. In this scenario, searching is reduced to a simple string matching problem or an SQL query. However, given the amount and diversity of mocap generated by even the smallest project, manual annotation schemes quickly become laborious, vague and unrealistic. Therefore, it is necessary to develop techniques to search massive mocap databases for a particular motion without the need for any prior annotation or labeling.

1.2 Motivation and Aims

Producing content for film, TV, animation, virtual environments or video games is a labour intensive process. This research project arises from the current trend the computer graphics community to simplify the production of visual content. This work endeavours to automate the production of animation whilst maximizing reuse of existing assets to reduce both cost and labour, yet maintain quality. The time and financial costs incurred in obtaining motion capture has prompted the development of tools to maximize their reusability. Due to the difficulties in dealing with human motion data, realistic and easy modification of motion capture remains an open problem. The goal is therefore to automate the editing of animations.

Our aim in this work is to simplify the editing of long motion sequences and large motion capture databases by allowing a single change to be applied over multiple motion instances. These instances would be found by searching for motions similar to that specified by the user.

Chapter 2

BACKGROUND

This chapter provides background information on motion editing. We start with an overview of motion capture and current trends in facilitating motion capture editing. The necessity for a motion matching framework is then motivated, and consequently, related work in both the computer graphics and time-series research fields is examined. An introduction to requisite concepts in time-series matching is also included. Since the scope of this work spans across a set of comprehensive research fields, the discussion is limited to papers directly related to the present goals.

2.1 Previous work in Motion Matching and Editing

Capturing human motion is expensive and time-consuming. This has prompted a surge of motion capture, (or *mocap*) related research [Gleicher 2001] in the last eight years to improve its reuse and applicability. A series of high-level techniques, outlined in Section 2.1.2, to modify and create motion capture have been introduced to circumvent the need either to recapture motions entirely or to alter them painstakingly by hand. After reviewing these techniques and identifying potential improvements, we explore existing work in motion capture matching which straddles two fields of research: computer graphics and time-series mining.

2.1.1 Motion Capture Format

Motion capture data holds a pose of the character at each instant in time and is roughly equivalent to a keyframe in traditional animation. This pose consists of values for all of the character's parameters at a given instant. Character models nowadays have hundreds and even thousands of degrees of freedom to achieve realistic hair, muscle-flexing and even the draping of clothes. But most motion capture systems and tools are focused on the *skeletal* rigid-body Degrees of Freedom (DoF); that is, the motions of a character that could still be observed even if all that remained was the skeleton. A character is therefore represented as a set of rigid segments (often called bones) that are required to remain connected. A skeleton (Figure 6) can be parameterized by the position and absolute orientation of one of the pieces, and the relative orientations between connected pieces (commonly called joint angles). The piece for which we specify position and absolute orientation is commonly called the root. The root segment of a skeleton (typically the hip/pelvis) is the only piece of a character for which we specify the position and orientation in absolute (world) coordinates. A bone hierarchy usually starts from the hips and branches down the legs, up the spine and to the arms. It is therefore convenient to represent the skeleton in a tree-structured hierarchy with the joints representing the edges and the bones represent the nodes (Figure 7).

Motion capture can be considered as a high-dimensional time-series where each dimension represents a different DoF of the character, and each time instant represents a keyframe of the animation.

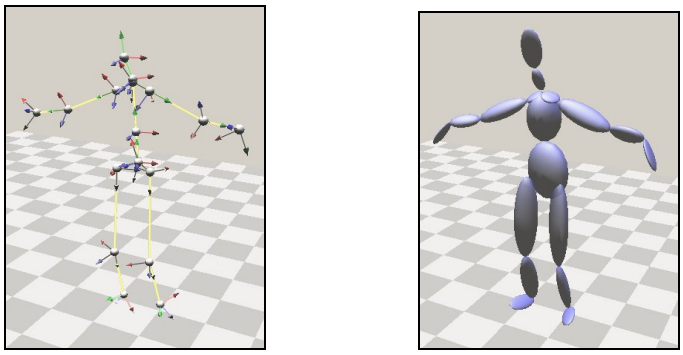

Figure 6: **Human Body Representation**. (*Left*) Bone hierarchy showing relative orientation axis at each joint. (*Right*) Corresponding pose visualisation rendered using ellipsoids.

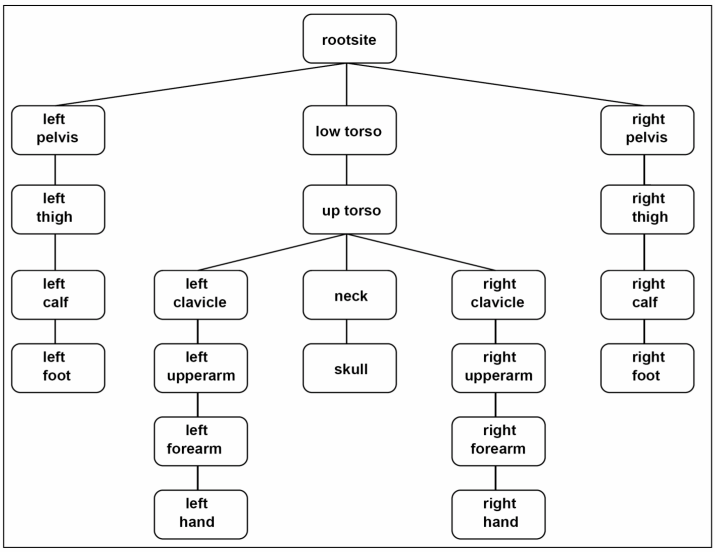

Figure 7: Example tree-structured hierarchy of human figure

Motion is commonly stored in ASCII files. In this research, we use the standard Biovision hierarchical data (BVH) file-format [Meredith and Maddock 2000], although our techniques support other skeletal formats. A BVH file is comprised of two sections. The first defines the hierarchy, bone lengths and initial pose of the skeleton (also known as base pose). The second section contains the motion as numeric data for each key-frame over time. The data consists of the global coordinates of the hips, the global orientation of the body and the local rotation between connected joints (i.e. bones). A set of three Euler angles define the joint rotations (Figure 8).

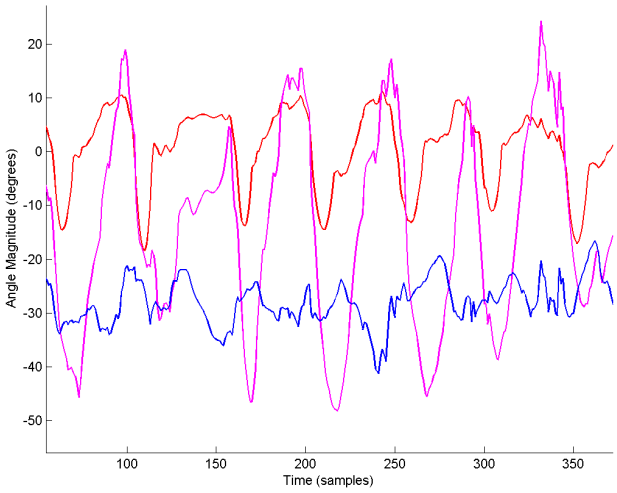

Figure 8: **Motion Capture Sample**. Three curves taken from a walking motion. Each curve represents a joint's Euler Y angle on the character.

2.1.2 Motion Capture Editing

This section examines a series of high-level techniques introduced to modify and create motion capture to circumvent the need for manual alterations or to entirely recapture motions.

Motion signal processing, introduced by Bruderlin and Williams [1995], uses an assortment of standard signal processing tools to edit motion curves. For example, we can also alter the range of the motion using motion warping, where a specifically generated displacement map is blended-in without disturbing continuity and global shape in the original motion (also proposed in [Witkin and Popovic 1995]). An example is depicted in Figure 9.

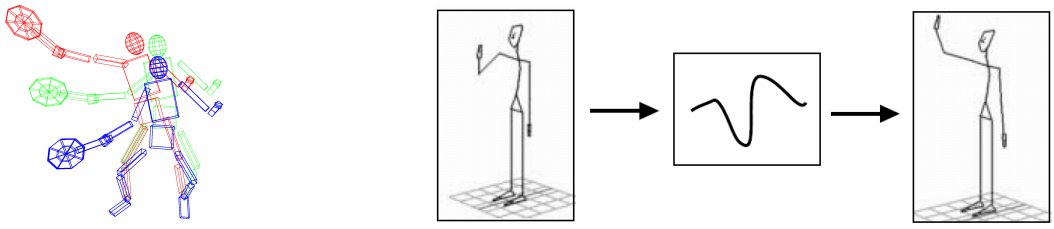

Figure 9: **Motion Warping Examples**. (*Left*) By changing only the ball impact keyframe in the forehand shot motion (green), a variety of warped sequences can be produced (red and blue). (*Right*) For modifications spanning multiple keyframes, a spline curve is fitted through user keyframe displacements for each modified degree of freedom. These curves are then added to the original motion. In this example, this results in a smoothly amplified waving motion.

As noted by Unama et al. [1995], we can scale different frequency components to affect the emotional content of motions. Filter banks can also be used to divide a motion signal into a number of components, which can be manipulated independently, and then reassembled to create a new motion signal (Figure 10). A simple observation, for example, is that high-pass filters can make movement appear more nervous or exaggerated and low-pass filters result in sad or sluggish perceived behaviour. High frequency stochastic noise is directly added to the motion to simulate nervousness in [Perlin 1995]. In a similar fashion, multi-resolution editing techniques [Lee and Shin 1999], which manipulate each different level of motion details separately, allow for motion enhancements/attenuations, blending, stitching and noise removal to be performed. Amaya et al. [1996] alter existing animation by extracting emotional transforms from example motions, such as a non-linear timewarp and a spatial scaling operation, which are then applied to other motions. Costa et al. [2000] extend this work by using the psychological principles of Laban Movement Analysis to enhance the style of mocap. They introduce two new groups of motion transforms referred to as *Shape* (shape flow, directional movement, and shaping) and *Effort* (space, weight, time, and flow) (Figure 11). Polichroniadis [2000] put all these motion editing techniques together to form a transform space which is empirically sampled to produce novel stylistic transforms of motion.

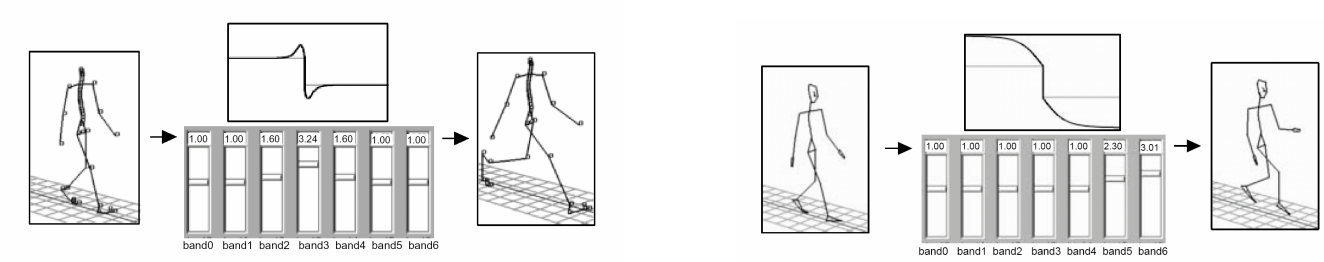

Figure 10: **Frequency band rescaling of motion**. By adjusting the gains of each frequency band for joint angles, we can vary the character of the new motion. In both diagrams, a frame from the original motion is given on the left and the corresponding frame from the filtered motion is shown on the right. The sliders in the middle dialog control the frequency response of the filters.

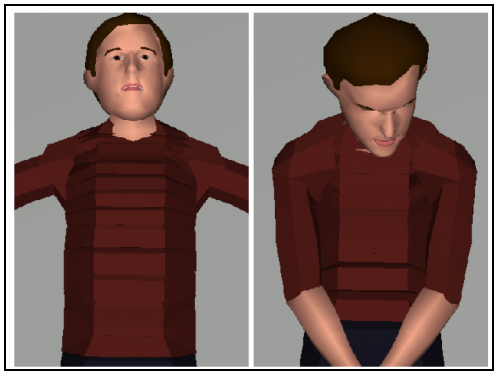

Figure 11: **Varying Shape Parameter of a Torso**. (*Left*) The torso *Horizontal* and *Sagittal Shape* parameters are increased to *advancing* and *rising* respectively. (*Right*) They are then reduced to e*nclosing* and r*etreating* qualifiers.

Infinite impulse response (IIR) filters can smooth, overshoot, or add wiggle to motion as first shown in the Inkwell system [Litwinowicz 1991] (Figure 12), with the result effectively changing the expressiveness of motion [Lasseter 1987]. Furthermore, non-linear filters such as the wave-shaping filter [Bruderlin and Williams 1995] can be used to introduce extra undulations (Figure 13). Time-warping [Bruderlin and Williams 1995] is useful to reparametrize the time of the motion to shorten or lengthen a motion (Figure 14). These transforms can introduce artefacts in the motion such as when the character's feet move when they should remain planted. These *footskates* can be efficiently eliminated using work developed by Kovar et al. [2002].

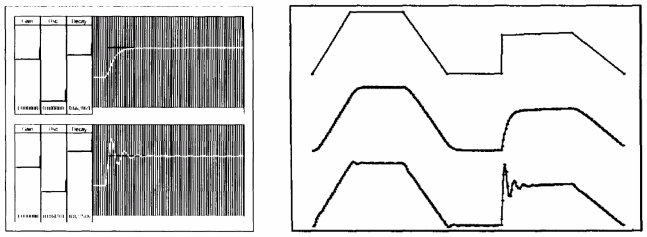

Figure 12: **Motion Filtering**. (*Left*) Two of example IIR filter step-responses. (*Right*) Both filters are applied to top motion curve

Figure 13: **Joint Angle Capping**. Joint angles of the arms are mapped via a shape function to introduced hard limits (*Left*), or softer limits to add undulations (*Right*).

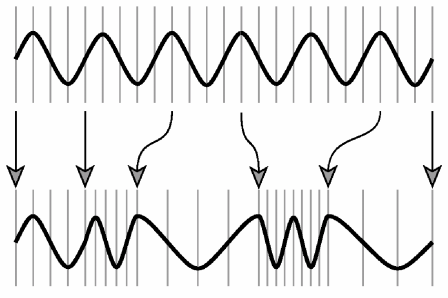

Figure 14: **Time Warping**. The original signal is compressed and expanded between specified points.

Motion blending is another useful editing operation. Bruderlin and Williams [1995] addressed the issue of blending two motions by first finding points of correspondence using dynamic programming to improve the quality of the linear joint curve interpolation. This is extended to multi-target blending where blending constraints, used to avoid unnatural motions such as conserving footplants or timing, are either specified manually [Rose et al. 1996; Wiley and Hahn 1997; Rose et al. 1998], semi-automatically [Ashraf and Wong 2000] or fully automatically [Kovar and Gleicher 2003].

A property of most motion editing methods that ignore inherent dynamics is that while they can effectively transform motion by small amounts, larger deformations reveal undesirable, unrealistic artefacts. A number of space-time constraints approaches [Gleicher 1997; Gleicher and Litwinowicz 1998; Rose et al. 1998; Lee and Shin 1999] address the need for both realism and controllability of character motion. In the space-time editing paradigm, the user first specifies pose constraints that must be satisfied by the resulting motion sequence (e.g. the character pose, such as a footplant, a crouch or a grabbing motion, at a particular keyframe in the animation). In addition to these constraints, the user also specifies an objective function which is a metric of performance or style (such as total power consumption of all of the character's muscles). The algorithm takes this space-time specification and finds the motion trajectories that minimize the objective function while satisfying the constraints (Figure 16). A space-time algorithm is also used in [Gleicher 1998] for motion retargeting which is a method for remapping captured motion onto drastically different characters (Figure 15).

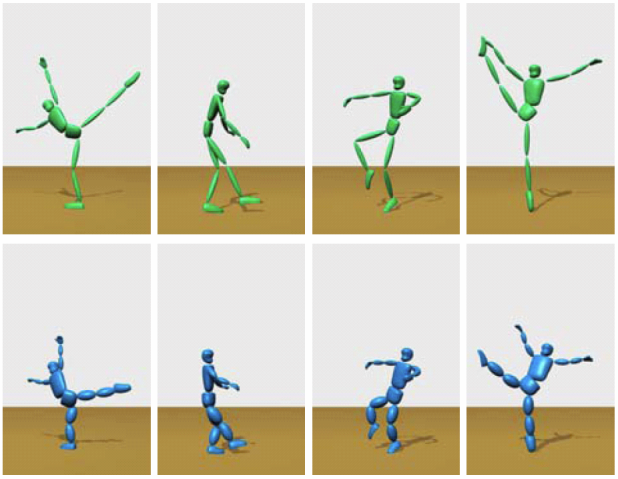

Figure 15: Motion Retargeting.

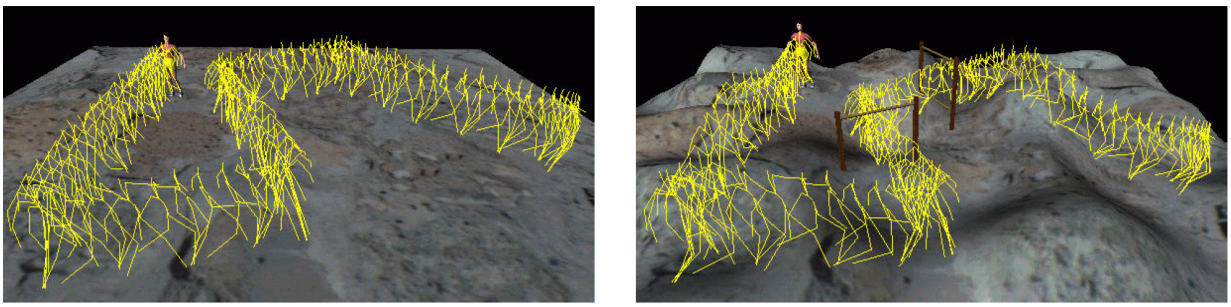

Figure 16: **Constraint-based editing**. (*Left*) The original walking motion on a flat surface. (*Right*) The same motion adjusted on a rougher surface by specifying appropriate ground constraints. Notice how the foot constraints are altered to accommodate the new terrain.

Since most of these methods may produce results violating the laws of mechanics, an editing method maintaining physical validity was first introduced by Popovic and Witkin [1999]. Their algorithm reduces the character to an abstract character which is highly simplified and only contains a reduced set of degrees of freedom that are useful for a particular animation. The edition and modification are then computed on this simplified character and mapped again onto the end-user skeleton. This work was recently made more computationally efficient in [Lui and Popovic 2002; Fang and Pollard 2003]. Similarly, motion balance filtering [Tak et al. 2000] can be used to quickly correct multiple unbalanced motions to balanced ones while preserving their original motion characteristics as much as possible. This is done by detecting and rectifying keyframes with centres of gravity outside the valid supporting area (Figure 17).

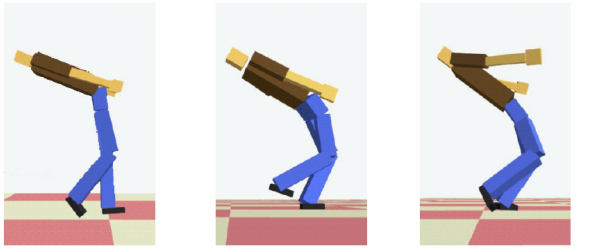

Figure 17: **Motion Balancing**. The original implausible motion (*left*) is balance filtered to produce a more realistic motion (*middle*). For comparison, a motion captured version of the targeted motion is shown (*right*).

All these approaches are generally designed to act on a single motion sequence at a time and would benefit from automated replication over multiple sequences. To our best knowledge, no system addresses this issue for mocap. Such a system would be conceptually close to Brooks and Dodgson's [2002] system where image editing operations are replicated over similar areas of an image (Figure 18), as opposed to mocap segments

in our work. Similar areas are located using the Euclidean distance between the selected pixel's neighbourhood and all other neighbourhoods in the image. For the case of motion capture, more efficient and flexible matching techniques are required to find similar motion sequences. In the following section, we examine existing approaches for motion capture matching in the computer graphics literature.

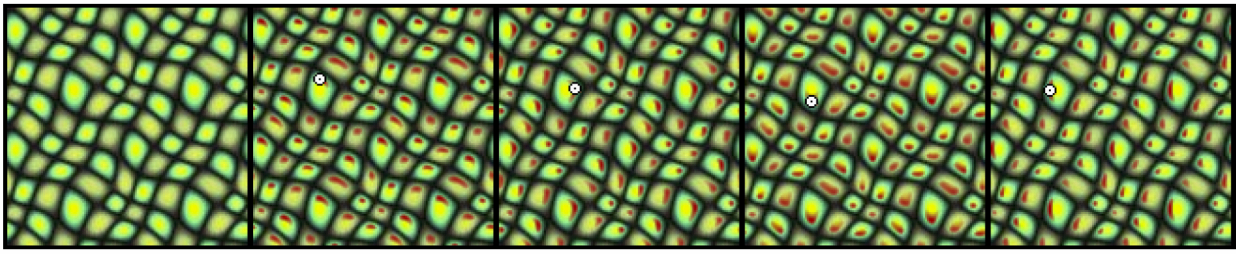

Figure 18: **Self-Similarity Based Texture Editing**. Given the original texture (*leftmost*), the user selects an area of the image and alters its colour or brightness. The editing operation is made to affect all areas that exhibit similar local neighbourhoods.

2.1.3 Motion Capture Matching

Recent work in human motion synthesis (Figure 19) has featured efficient search algorithms over mocap databases. However, these search algorithms are geared towards *generative output* in that exact matches are not found, but created on-the-fly. A set of positional, temporal or stylistic constraints are defined by the user and used to guide the synthesis. This is done by first pre-processing the mocap database to segment, and compress and hierarchically cluster either individual frames [Lee et al. 2002; Molina Tanco et al. 2000] or short mocap sequences [Arikan and Forsyth 2002; Kovar et. al 2002; Sidenbladh et al. 2002]. This forms a search index which is used to determine the best transitions between each pose or short mocap sequence employed during mocap synthesis. The mocap segment which fits the current user constraint and which has the highest probability of occurring after the current segment, is appended to produce a continuous stream of motion. Matching therefore typically occurs at the frame-to-frame level where the end frame of a currently synthesized segment is compared to the start frame of other segments in the index. These approaches do not address the fundamentally different problem of sequence-to-sequence level matching, where matches are found across time between a series of frames. In their current form, they are not designed for finding exact matches. Also, their pre-processing phase requires that matching be carried out over a fixed body area. Hence, matching areas cannot be dynamically altered at runtime, nor can the utilized similarity measure and matching weights be adjusted. In addition, updating the search index usually requires a complete re-run of an extended pre-processing phase. Extending these systems to mocap search-by-query might be feasible but is not a trivial undertaking.

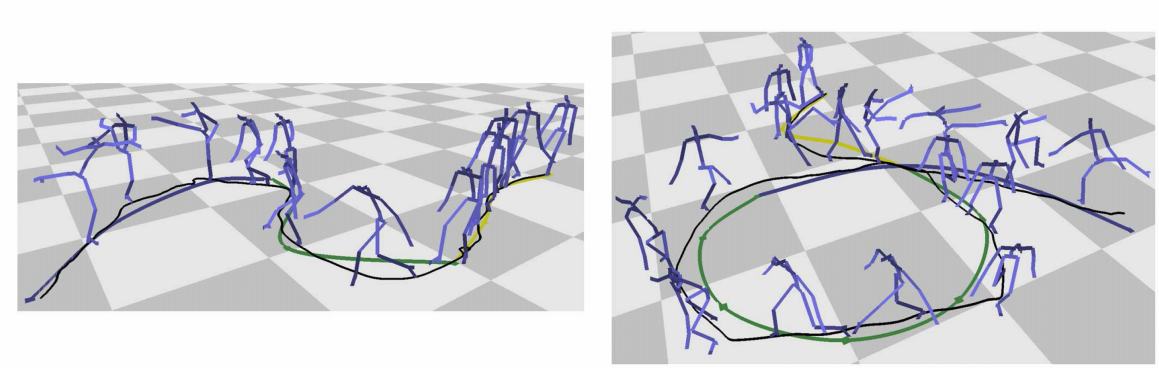

Figure 19: **Motion Synthesis**. These motion sequences are synthesized to fit path and stylistic constraints. The character is required to follow the coloured path with a walking motion, a sneaking motion and finally with kicking motions. The desired transition points are indicated by the curve colour changes. The black curve shows the paths travelled by the character in the synthesized motion.

Pullen and Bregler's [2002] motion texturing method used mocap as a source for augmenting partially key-framed motions with high-pass band information and adding degrees of freedom (Figure 20). Their method relies on a mocap search mechanism where the query motion, defined by any DoF (degree of freedom), is segmented along first derivative changes and the closest matching segments for each limb at each instant are found. However, their method does not operate in real-time and does not scale since it was designed for non-real-time operation over a single mocap sequence, not a mocap database.

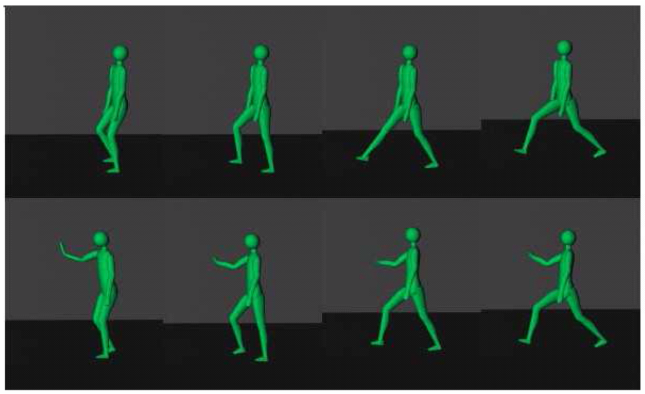

Figure 20: **Motion Texturing**. (*Top*) The original motion created by manually specifying a few keyframes. (*Bottom*) Motion capture is then used to enhance the animation and make it more lifelike. This is done finding matching regions between the keyframed motion and a mocap dancing sequence, and by pasting high frequency detail from the mocap at corresponding matching regions.

2.1.4 Related Time-Series Research

Due to the apparent lack of relevant work in motion capture matching from the graphics community, we look at methods emanating from the field of time-series pattern matching. The problem of efficiently finding patterns in massive time-series databases has attracted great interest in the database and data mining communities in the last decade [Agrawal et al. 1993; Hetland 2003]. Indexing of time-series has concentrated most of the attention in the research community, and this can partly be attributed to the explosion of database sizes. If we would like to allow the user to search these vast databases, we should organize the data in such a way, so that one can accurately and efficiently retrieve the data of interest. An introduction to the core concepts in time-series mining is given in Section 2.1.4.1 , followed by a discussion of all relevant approaches applicable to motion capture matching in Section 2.1.4.2 .

2.1.4.1 Time-Series Matching Preliminaries

Time-series

Motion capture is characterised by sequences of recorded values, with one such sequence for each joint. These values, representing rotations or positions, are real numbers recorded at regular intervals dictated by the capture sampling rate. This is a typical example of a **multi-dimensional time-series** for which many matching techniques have been developed for data-mining. They typically deal with the problem of similarity search in massive time-series databases, where given a time-series query, we find all the time-series in the database that are similar to the query. This is also called **query-by-example**. Similarity searching is not a trivial problem. The two primary difficulties are defining a similarity measure and time complexity.

Metric versus Non-Metric Distance Measures

Similarity is one of the central bases of information retrieval. To find content that is similar to a query, an information retrieval system must quantify similarity accurately and retrieve similar content efficiently. A distance function $d(x, y)$ is used to measure the similarity between pairs of objects *x* and *y*. It is defined as a metric distance if it satisfies the following conditions:

- $d(x, y) = 0 \quad iff \quad x = y$
- $d(x, y) > 0 \quad for \quad x \neq y$

- $d(x,y) = d(y,x)$ (*symmetry condition*)
- $d(x,y) \le d(x,z) + d(z,y)$ (*triangular inequality condition*)

For example, the widely used Euclidean distance detailed in Section 3.2.3.2 , is metric. However, it has been verified that the Euclidean distance (more generally, metric distances) between objects is not the best way to model perceptual similarity [Murphy and Medin 1985; Tversky 1977]. Psychology research suggests that human similarity judgments are non-metric and dynamic [Medin et al. 1993]. A non-metric similarity function is typically formulated only after the objects are compared, not before the comparison is made; and the elements for the comparison are dynamic. Distance functions that are perceptually accurate, and consequently robust to noisy data, will typically violate the triangular inequality (see Figure 21). There are many non-metric variants utilized in DNA matching and image retrieval; so preference is given to those that deal effectively with outliers, support sequences of different lengths and are relatively efficient to calculate. The problem with non-metric distances is that they are difficult to index with standard indexing techniques which rely on the triangular inequality property.

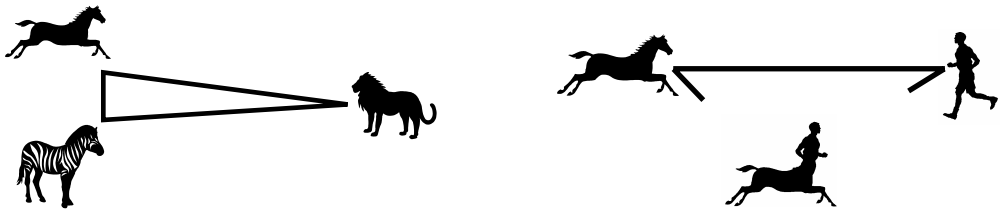

Figure 21: **Triangular Inequality**. (*Left*) The triangular inequality requirement maps nicely onto human intuitions: the horse and the zebra are both rather similar, and both are very unlike the lion. (*Right*) Sometimes the triangular inequality requirement fails to map onto human intuition. The horse and the man are very different, but both share many features with the centaur. If we used the shared attributes of the man and centaur to compare the man and the horse, the latter pair might not seem similar, and vice versa. This relationship does not obey the triangular inequality since a distance function using a fixed set of respects cannot capture objects that are similar in different sets of respects.

Most of the related work on multi-dimensional time-series has concentrated on the use of the L_p Norm family of metrics [Roddick and Hornsby 2000; Kim et al. 2001]. The advantage of these simple metrics is that they allow efficient indexing by dimensionality reduction techniques. The Minkowski norm distance between *X* and *Y*, when their lengths are equal, is defined as:

$$L_N(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^N \right)^{\frac{1}{N}} \quad \text{with} \quad N \in \Re \quad \text{and} \quad X, Y \in \Re^n$$

where $\Re$ is the set of real numbers. For *N*=1 it is the Manhattan L_1 metric and when *N*=2 it is the more popular Euclidean L_2 metric. There is also the L_∞ distance which is equivalent to focusing the similarity

on the maximum found difference over all elements. This is not particularly meaningful over mocap data. Peng et al. [2000] introduce a similarity measure that calculates the distance over only a subset of values from a time sequence, which are the peak points (i.e. the local minima and maxima). However, the peak points, or *landmarks*, of different time sequences may correspond to different time points. It is also not clear if landmarks are meaningful over multi-dimensional data and if they can emulate Dynamic Time Warping (DTW) [Chu et al. 2002] which has well-documented success in real-world applications. In the non-metric DTW, we are allowed to extend each sequence by repeating elements before calculating the distance between the extended sequences. This allows a degree of acceleration-deceleration along the time axis. Warping is therefore preferred whilst matching over spatio-temporal trajectories. Because of its well-documented lethargy, this technique was deemed impractical for large databases until a recent paper demonstrated that DTW can be indexed with no false negatives [Keogh 2002]. Another non-metric distance function similar to DTW is the Longest Common Subsequence (LCSS) measure which is a variation on the edit distance [Bozkaya et al. 1997]. LCSS is more robust to noise than both Euclidean and DTW; it can also be efficiently upper-bounded [Vlachos et al. 2002]. DTW and LCSS are described in further detail in Sections 3.2.3.3 and 3.2.3.4 respectively.

Whole Matching and Subsequence Matching

There are two categories of similarity queries in time-series databases: **whole sequence matching** and subsequence matching. In whole sequence matching, it is assumed that all sequences to be compared are of the same length. The answer to a whole sequence similarity search is all the time-series in the database, whose distance from the query are less than a given threshold. The distance is evaluated using a metric or non-metric distance measurement function.

In **subsequence matching**, the time-series in the database can have different lengths. The lengths of these candidate time-series are usually larger than the length of the query time-series. The answer to a subsequence query is any subsequence of any candidate time-series whose distance from the query is less than the given threshold. This research is focused on subsequence matching since the query motion and database candidate lengths are not fixed. Note that whole sequence matching is a special case of subsequence matching. The obvious brute force solution to subsequence matching is to slide the query motion against every longer database sequence and record the distance at each point. This is called **sequential scanning** (also referred to as: linear scan or brute force) and scales poorly because the computing time increases linearly with the size of the database (Figure 22). Also, linear scan requires that the whole data be loaded into main memory. Thus, a large number of disk accesses are required.

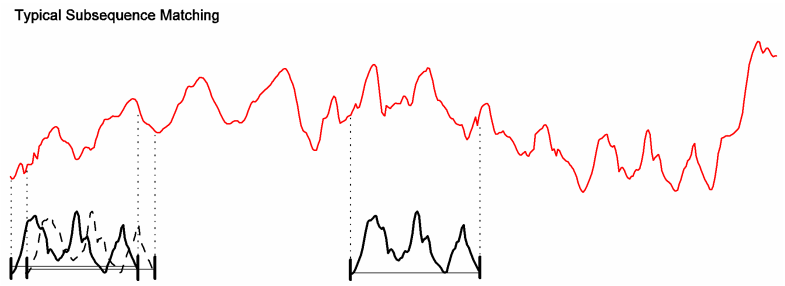

Figure 22: **Linear Scan**. Typical brute force subsequence matching using a sliding window. The diagram shows the query in black at different offset positions along the data in red.

k Nearest Neighbours (k-NN)

Unlike a traditional relational database system where a selection query consists of a set of precise selection predicates and the user expects to get back the exact set of objects that satisfy these predicates, in ***k*-NN** queries, the user specifies target values for certain attributes and does not expect exact matches to these values in return. Instead, the result to such queries is typically a ranked list of the *top k* objects that best match the given query, where *k* is the number of matches desired. An alternative to *k*-NN is the **range query** where we find all candidate sequences within a threshold distance error of the query.

Indexing

Many indexing structures have been proposed to solve the complexity problem. An index is a data organisation structure that allows fast retrieval of the data. There are two categories of indexing methods. The first is to consider each time-series as an *n*-dimensional tuple and use a high-dimensional index structure to index the tuples. This structure is called a **spatial access method**, such as the R-tree [Guttman 1984], where the high-dimensional space is partitioned with hierarchically nested, and possibly overlapping, boxes. Tuples are indexed in each box which intersects them. The objective is to search only those boxes that could potentially contain good matches and avoid everything else. The problem is that they require the use of a metric distance because the triangular inequality must hold and generally cannot be extended to handle subsequence matching. They also suffer from the curse of dimensionality which decreases their performance over high-dimensionalities. This is because the volume of the boxes that enclose the dataset's *n*-dimensional tuples grows exponentially with *n*. This affects the tightness of the space partitioning, which in turn affect its usefulness.

Alternatively, **dimensionality reduction** techniques can be used to approximate the data into a lower dimensional space where similarity can be evaluated efficiently [Faloutsos et al. 1994]. The *n*-dimensional tuples that represent the time-series are projected to a *k*-dimensional space, with $k<<n$, that preserves their distances as well as possible (Figure 23).

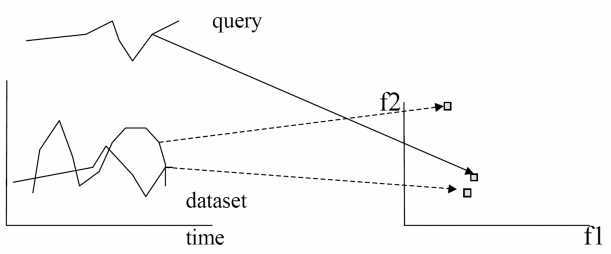

Figure 23. **Dimensionality reduction**. The query (*top-left*) is mapped to a new space along with the database sequences (*bottom-right*). We find the nearest neighbours to the query in this new space (*right*). We then retrieve the original sequences from the database, compute the true distances and keep the true closest neighbour.

This approximation of the data should fit in main memory yet retain the essential features of interest. The task is approximately solved in main memory so that hopefully only very few accesses to the original data on disk are necessary. Only a limited number of costly full calculations are then required to confirm the tentative solutions. Since the entire database is not examined, two potential problems can occur: **false alarms** (i.e. false positives) and **false dismissals** (i.e. false negatives). False positives are when the approximate query answers include some time-series in the database that are actually not qualified answers. Since false alarms can be removed, or pruned, by confirming distance estimates on the original data, they only affect the matching speed and can be tolerated so long as they are relatively infrequent. With false negatives, the approximate query answers do not include some time-series in the database that are actually qualified answers. False negatives are unacceptable since they affect correctness, whereas false alarms affect only time. Since correctness is more important in our research, we use an index with the no false-negative property. This is guaranteed by enforcing that distance between sequences under the dimensionality reduction function T lower-bounds the original distance between time-series such that $dist(T(x),T(y)) \le dist(x,y)$.

2.1.4.2 Related Work in Time-Series Matching

Time-series Subsequence Matching

Whole sequence matching over one-dimensional time-series has been the principal focus of the data-mining community. The first solution to this problem was developed by Agrawal et al. [1993] where they transformed time-series into the frequency domain using the Discrete Fourier Transform (DFT). The first *f* DFT coefficients were then used to map each sequence into an *f*-dimensional feature space. A spatial access method (e.g. an R-tree [Guttman 1984]) is used to organize the resulting feature vectors. Given a query sequence, they retrieve the closest points. These are only approximate answers since they include the points that are not the correct answers. The false alarms are then discarded using sequential scan.

There have been surprisingly few methods for dealing with the more complex issue of subsequence matching. Faloutsos et al. [1994] generalize the above method to subsequence matching. This first step is to copy and generate subsequences using a fixed-length sliding window. Next, they reduce the dimensionality of these copy sequences using DFT to make *f*-dimensional vectors. By dividing data sequences into sliding windows, this generates too many points to be stored individually in an index. Since a trail of these *f*-dimensional vectors is formed, minimum bounding rectangles (MBRs) are used to divide the trails into sub-trails that contain hundreds or thousands of points. Once again, an R-tree is used to organize the MBRs. The query sequence is broken up into disjoint windows and matches for each window are retrieved from the R-tree and accumulated to form the candidate set. False positives are then pruned by calculating the true distances. The problem is that many false positives are generated, degrading performance. Storing MBRs only causes false positives by not allowing point-to-point comparison in the index level for checking distances. Moon et al. [2001] address this problem by instead breaking up the data sequences into disjoint windows and a query sequence into sliding windows. This reduces the number of window points to store and the number of false alarms shrinks, improving performance. Moon et al. [2002] later generalized the approach by allowing a larger window to reduce the number of points while reducing the number of false positives. Lee et al. [2000] further extended Faloutsos et al.'s [1994] original ideas to support subsequence matching over multi-dimensional sequences. Unfortunately, it allows false negatives as noted in [Kahveci and Singh 2001]. The later authors introduce subsequence matching over multi-dimensional time-series with support for shift and scale invariance in [Kahveci et al. 2001]. However, with motion capture, we do not want to ignore scale and shift invariance because that is what characterizes a mocap sequence from another. Also, the memory overhead of their index requires space multiple times the size of the original dataset.

There are several drawbacks with these methods. When the length of a query exceeds the size of the window, performance degrades sharply. In order to cope with long queries or to represent original data sequences faithfully, higher dimensionality vectors must be used. However, the performance of the spatial access methods degrades as the number of dimensions increase [Weber et al. 1998]. Conversely, if low dimensionality window vectors are used to avoid this phenomenon, too many false alarms are generated as the vectors are likely to become an inaccurate representation of the original data sequences. More fundamentally, these methods cannot process queries whose length is smaller than the window size.

Another assumption made in all these methods is that the similarity between sequences is metric and Euclidean. Since DFT is used for feature extraction, only the Euclidean distance between sequences is preserved. This assumption prevents matching between sequences of different lengths or different sampling rates. In recent years, there has been an increasing awareness that the Euclidean distance is inappropriate for many real world applications. The limitations of Euclidean distance stems from the fact that it is very sensitive to distortions in the time domain [Chu et al. 2002] which are common in human motion. The reason for this is that if the triangular inequality does not hold, most indexing techniques such as spatial access methods do not work. To the best of our knowledge, there are only two indexing methods for subsequence matching using the non-Euclidean distance measures.

The first approach by Park et al. [2000; 2001] supports indexed subsequence matching with the dynamic time-warping (DTW) distance measure. Sequences are divided into piece-wise linear segments where each segment is then converted into a symbol. The symbols are then indexed in a suffix tree, as this index structure does not assume the triangular inequality (which does not hold for non-metric measures such as DTW). Several difficulties arise in this approach. First, the search times increase quadratically with the average length of data sequences. Secondly, the index is one to two orders of magnitude larger than the data itself making such large space overhead untenable for large databases. In any case, the claimed speed up is rather modest and there are no false negative guarantees [Keogh 2002].

The second method by Morinaka et al. [2001] also works by approximating the sequences into piece-wise linear segments. The Manhattan L_1 metric is used as a similarity measure. The index is used to find approximate answers by sliding the approximated query along the index with no false negatives. Since the query and sequence are small, calculating the approximate answer is faster than sequential scan. False positives are then pruned by calculating the true distances on the data itself. Since the Manhattan distance is

used, there is no support for non-aligned time-series. Likewise, sequences of unequal length or sampling rate cannot be compared.

None of these methods support all the essential features required for a viable motion capture matching system. These include support for:

- Multiple distance measures to allow for different notion of motion similarity.
- Scale-invariance, to find matches at different speeds.
- Multi-dimensional time-series with the possibility of dimensionality weighting, so that different joints can be given more or less importance in the similarity calculation.
- Subsequence matching, to support query motions of any length.
- No false-negative, so that no matches are missed.

To our best knowledge, there is currently no equivalent in the literature.

Time-series Matching Applied to Motion Capture

There have been several related attempts at tackling retrieval-by-content over human motion capture. Osaki et al. [1999; 2000; Shimada and Uehara 2000; Mori et al. 2002] propose a system where the motion data is divided into segments by detecting velocity changes. These segments are then clustered and converted into symbols assigned to each cluster centre. They refer to these as *primitive motions*. Under this scheme, a motion sequence becomes a sequence of primitive motions represented by a series of symbols (Figure 24). The matching is then done using standard string matching techniques.

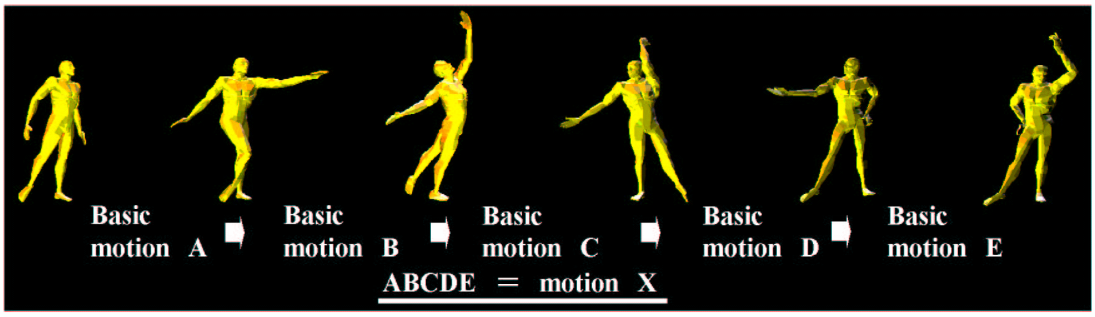

Figure 24. Motion represented as a combination of basic motions.

The problem is that this approach has no fundamental basis for dividing time-series (as the authors concede in their later work [Tanaka and Uehara 2003]). In their more recent work [Tanaka and Uehara 2003], a system is developed to automatically extract recurring patterns, or *motifs*, from a small mocap database. To this end, they map multi-dimensional positional mocap data to a one dimensional time-series corresponding to the largest eigen-valued eigenvector using Principal Component Analysis (see Section 3.2.7.1 for details on PCA) where most of the information resides. The one-dimensional curve is then converted to a sequence of symbols using Piecewise Aggregate Approximation (PAA) [Lin et al. 2002] (Figure 25) from which the motifs are extracted. Although the system is used for motif extraction, query-by-example can also be carried out on it. The one-dimensional PCA approximation, compounded with the PAA symbolisation, means that only gross pose information in the motion is retained. It is not clear from the paper if the utilized positional data included global translation of the body, or is relative to a local body coordinate frame. In the former case, the PCA would be dominated by the global translation component minifying the impact of body pose differences (see Section 3.2.7.3 for details). Ultimately, the scheme is fundamentally flawed since recent work by Keogh et al. [2003] proved that rule-based extraction techniques, as well as clustering, over subsequences is meaningless: this is because the output of the system does not depend on the input. This reinforces our choice not to pursue the subsequence clustering approach in our research.

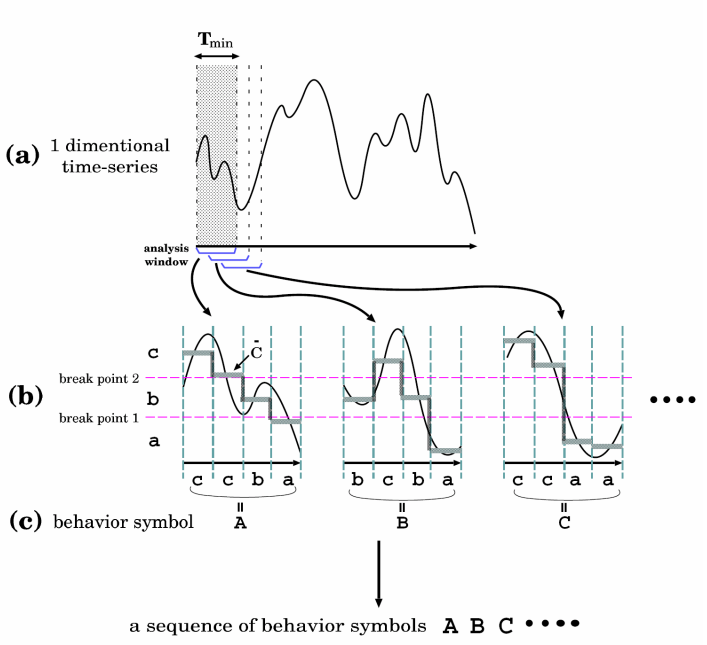

Figure 25. **Visualisation of segmentation of the PCA compressed mocap data**. (a) Subsequences are obtained by shifting the analysis window. (b) Each subsequence is transformed into a sequence of PAA symbols using PAA. (c) A sequence of symbols, or behaviour symbols, is assigned for every pattern in the order of PAA symbols.

Gesture Recognition Research

The analysis of human motion is also relevant in computer vision. A vision-based gesture recognition system is typically composed of three phases [Pavlovi'c et al. 1997]: gesture modelling, gesture analysis and gesture recognition. The model may consider spatial and temporal parameters depending on the application. Spatial parameters include the pose and the motion in space. The analysis phase extracts model parameters from images (Figure 26). This phase is followed by gesture recognition in which the parameters are classified and interpreted to infer the observed gesture.

Figure 26. **Computer-vision based motion recognition**. The motion of a 3D human model is extracted from a sequence of images.

A variety of techniques have been used to this end such as neural networks [Huang et al. 1998], temporal models for gesture recognition [Bowden and Sarhadi 2000], spatial modelling of gestures [Davis and Shah 1999], recognition of gestures using hidden Markov models (HMM) [Starner and Pentland 1995], Principal Component Analysis [Moghaddam 1999] and position-based gesture recognition [Ng and Ranganath 2000]. The reader is referred to the survey by Moeslund and Granum [2001] for more details. Due to the difficulty of extracting 3D positional data from 2D sources, the core of the research focuses on the motion extraction phase.

The problem tackled by these approaches is fundamentally different from that of our targeted motion capture matching system. In the above methods, the recognition phase reduces to a classification of temporal data. There is a training phase where a model of the gesture to recognize is learnt from multiple examples. The goal is then to classify an unseen gesture amongst the pre-learnt gesture models. The set of gestures to recognize is comparatively small, in the order of 10 to 100 gestures [Starner and Pentland 1995]. In query-by-content, we cannot feasibly build an index containing a learnt model for every possible subsequence of any length in the mocap database. The size and computation time for the index would be untenable. The techniques described earlier in time-series matching are more suitable for our goals.

2.1.5 Conclusion

In this section, recent research in graphics and time-series related to the automated motion editing goals of this work was investigated. In graphics research we noted several potential problem areas, including the lack of work on repeating motion capture editing operations over multiple motions, and the related problem of finding relevant matches. In time-series research we found that no existing approach completely fulfils our requirement of a fast, flexible and compact matching algorithm utilizing distance functions adapted to motion capture. There is clearly a need for such techniques, which we endeavour to address in the remainder of this report.

Chapter 3

MOTION MATCHING AND EDITING

In this chapter, the problem of simplifying the editing of motion is addressed. The possibility of storing large quantities of motion capture motivates the present research. We exploit the inherent structure of motion capture to realise a flexible and efficient matching algorithm. The notion of replicated editing is also introduced to allow editing operations over a single motion to be repeatedly applied over all matches with minimal user-input. In general, finding a similarity measure for two motions is not easy because sequences that are qualitatively the same may be quantitatively different. To answer this problem, an appropriate representation of the human body is devised as well as a set of configurable similarity measures.

3.1 Aims

The popularity of motion capture, or *mocap*, in computer animated movies and three-dimensional computer games has motivated the development of many techniques to tackle the difficult problem of motion editing. The existence of large libraries of human motion capture has resulted in a growing demand for content-based retrieval of motion sequences without using annotations or other meta-data. We address the issue of rapidly retrieving perceptually similar occurrences of a particular motion in a long mocap sequence or unstructured mocap database for the purpose of replicating editing operations with minimal user-input. One or more editing operations on a given motion are made to affect all similar matching motions. This general approach is applied to standard mocap editing operations. The style of interaction lies between automation and complete user control.

3.1.1 Matching Aims

Unlike recent mocap synthesis systems [Arikan and Forsyth 2002; Kovar et. al. 2002; Sidenbladh et al. 2002], where new motion is generated by searching for plausible transitions between motion segments, the goal here is to efficiently search for similar motions using a query-by-example paradigm, while still allowing for extensive parameterisation over the nature of the matching. The animator first selects a particular motion, by specifying its start and end times, and the system searches for similar occurrences in a mocap database (Figure 27). For maximum usability, the mocap matching algorithm must provide a fast response to user queries over extended unlabelled mocap sequences, whilst allowing for spatial and temporal deviations in the returned matches. The pre-processing phase, necessary to build up an efficient search index capable of guaranteeing no false negatives, can be non-real-time but should remain unsupervised.

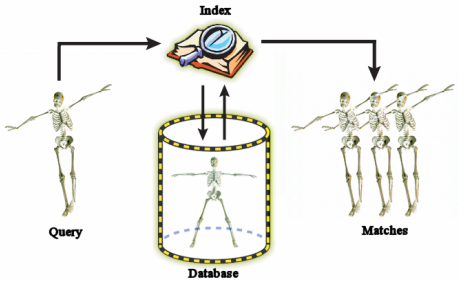

Figure 27: Matches to the user query motion are returned in real-time using an efficient search index, previously extracted from the motion database.

Matching should support a variety of similarity measures that cater for noisy motion with variations in the time axis to produce robust correspondences between keyframes. Using the same index, the system should allow rapid location of potential candidates within a dynamically user-defined temporal range; that is, matches that are up to k percent shorter or longer. The effect is that matches can be located independently of the speed at which the actor performed them.

The animator should have the ability to select the body areas used in the matching, so that, for example, all instances of a grabbing motion are returned, irrespective of the lower body motion. Negative matches should also be allowed to exclude particular motions in the returned matches. For instance, if the user selects a kicking motion as a positive example and a punching motion as a negative example, then only matches with a kick, without a simultaneous punch, will be returned. Finally, in the case where many potential matches exist, the query results should be clustered and identify a representative of each cluster. These should be displayed simultaneously so that the animator can rapidly dismiss undesirable classes of matches.

Given these requirements, we state the desirable properties that our indexing scheme should have:

- Be orders of magnitude faster than linear scan.
- Support kNN queries and range queries.
- Support subsequence matching over multi-dimensional time-series.
- Require small space overhead.
- Support subsequence matching for queries of any length.
- Support heterogeneous databases composed of mocap sequences of any length.
- Support dynamic updates such as insertions and deletions of motion sequences without incurring a complete rebuild of the index.
- Should be built in a reasonable amount of time in a pre-processing phase.
- Support multiple distance measures to accommodate for different matching constraints.
- Support scale-invariance so that matches within a certain stretch or shrink factor of the query are recognized.
- Should be *correct* in that it should return all qualifying matches without missing any i.e. there should be no false negatives.
- Minimize the number of false positives to reduce I/O costs to retrieve the original sequences for true distance calculations.

- Allow the dimensions over which matching is performed to change dynamically so that the body matching area can vary.
- Support dynamically configurable weights for each dimension, so that the user can weight different body parts independently.
- Support for a flexible similarity measure with time-alignment of compared sequences.
- Enforce a minimum distance between valid matches, to eliminate trivial matches.

3.1.2 Editing Aims

As well as being a practical and efficient mocap search mechanism, the system should allow an editing operation performed on the query motion to be repeated on all selected matches (Figure 28). A variety of motion editing operations should be supported such as time-warping, motion displacement mapping, motion-wave-shaping, blending, smoothing and various other linear and non-linear filtering operations. The index should support quick updates so searches including the newly edited matches, or completely new sequence inserts, are possible with minimal delay. Contextualized transforms should be supported to enable the original transformation performed on the query motion to be locally tailored before being applied to each match. This can be used to differentiate matches or to ensure that edits are meaningful and useful across all matches.

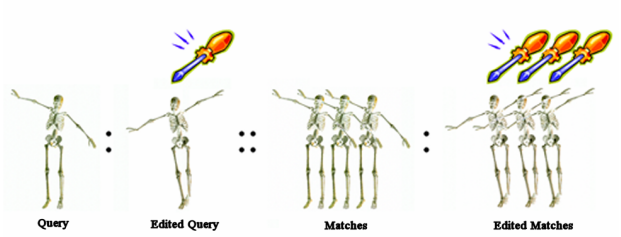

Figure 28: **Replicated editing**. The animator's motion edit performed on the query motion is automatically replicated over all found matches.

3.2 Motion Capture Matching

Having examined prior work in motion capture and time-series matching in Section 2.1, we now present our own matching system. Although it is designed with motion capture in mind, it is applicable to other multi-dimensional time-series. A general overview of the matching system is first given in Section 3.2.1.

In Section 3.2.2, we tackle the problem of finding a representation of the mocap data suitable for our matching purposes. In total, four parameterisations of the body joints are considered. Two of which are retained.

We then focus on the means by which we evaluate similarity between two motion sequences in Section 3.2.3. In Section 3.2.4, these measures are quickly estimated to avoid the need to calculate the true distance. By simply using our indexed data approximations, we can lower-bound the true distance without accessing the original data from the database. Next, in Section 3.2.6, the notion of scale-invariant matching is introduced and we demonstrate how it can also be bounded under our scheme. Finally, in Section 3.2.7, efficient whole-body matching is made possible by compressing the body representation using a linear dimensional reduction technique.

3.2.1 Matching Overview

The motion template to be matched can either be an existing mocap snippet or an estimate of the target mocap sequence sketched using a 3D animation package (Figure 29). The user then sets the matching area, the matching weights or thresholds depending on the distance function, and the number k of targeted nearest neighbours.

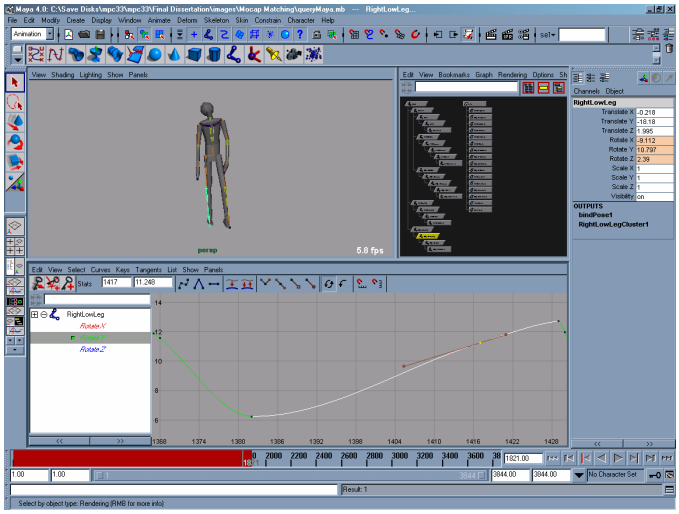

Figure 29: **Manual query definition**. The screenshot of *Alias|Wavefront's Maya* 3D animation package shows a step in the creation of a key-frame motion sequence. The animator specifies a set of poses for a sparse set of keyframes by either directly editing the character in 3D (*top-left*) or using the curve editor (*bottom*).

Linear search on a large multi-dimensional mocap database is not feasible with any reasonable degree of responsiveness. Our accelerated matching procedure begins instead by scanning a pre-constructed index. With our design, the index only produces a set of candidate matches. This step is termed the *filter step*. For each candidate approximation in the index, we estimate its minimum distance from the query. This results in a sorted priority queue of estimates (along with a pointer into the database) for each candidate. In the refinement step, for promising candidates, we then retrieve the exact motions from the database and calculate their true distance from the query. Since we start with our best estimate, the odds are our match will be at the top of the queue, consequently requiring only a limited number of secondary memory accesses and full distance calculations. The current best candidate is dropped as soon as we find a better one (i.e. one with a lower true distance). We stop cycling down the queue until we reach an estimate above our current best true distance. The operation repeats on the updated queue until the k nearest matches are found. Figure 30 depicts the overall process. We now describe how we obtain the index and look at the different matching steps in more detail.

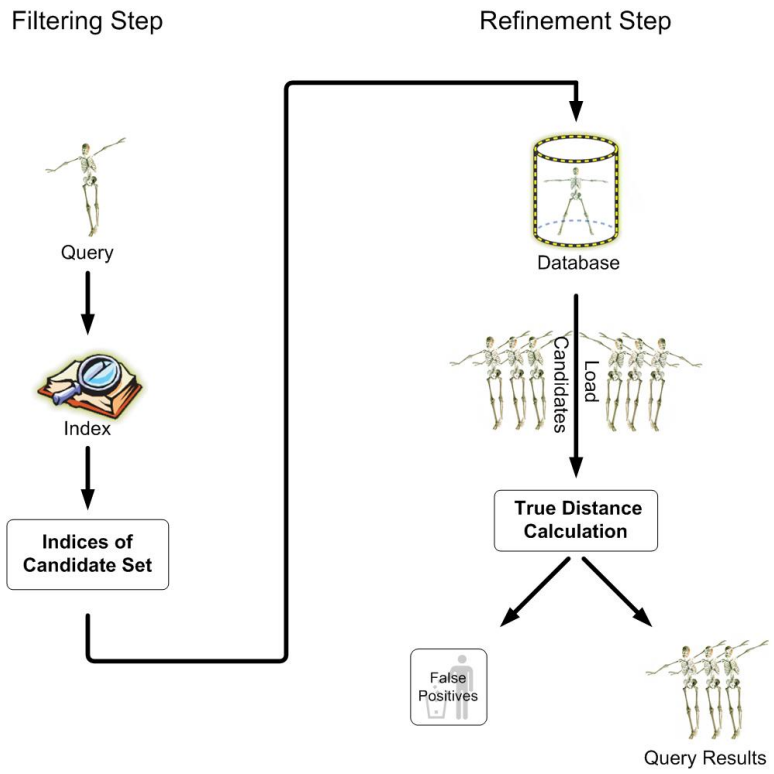

Figure 30: **Multi-step query processing**. Given a query motion, the index is scanned to produce a set of candidate matches, for which their exact motions are retrieved from the database and their true distance from the query is calculated. False positives are ignored and the top *k* nearest matches are presented to the user.

3.2.1.1 Index Generation

The goal of the index is to minimize the number of bytes it occupies whilst maximizing the accuracy of the sequence approximations for better time efficiency. The approximation is obtained by splitting the motion trajectories into Minimum Bounding Rectangles (MBR). An MBR is an *n*-dimensional rectangle bounding the indexed spatial data.

The nature of the index depends on the targeted matching area and support for scale-invariance. Depending on the body representation, the matching area is fixed or not. If we use the positional body parameterisation, which describes the body as a set of joint coordinates, the matching area is locked to whole body matching. If we prefer to interactively select the matching area, the angular body representation is more suitable. In this parameterisation, the body is described by a set of independent and local joint rotations. Basing the index on positional data means the matching area must be fixed but guarantees meaningful and efficient compressibility under Principal Component Analysis (PCA), a linear dimensionality

reduction technique. On the other hand, a rotational-based index ensures that any region(s) of the body can be added, weighted, or suppressed in the matching. This comes at the cost of diminished index compactness, as well as lower performance during whole body matching.

To find matches that are up to $\pm s$ percent shorter or longer than the query, scale-invariance within a pre-defined range is achieved by bounding each database trajectory by its *scaling envelope*. The envelope represents the original trajectory under all stretching conditions and it correspondingly affects the distance estimates. The enveloping stage comes before the MBR segmentation so that the MBRs enclose both the original trajectory and its scaling envelope.

Highly heterogeneous mocap databases, consisting of many short motions, many long sequences or an assortment of both, are supported by our index. Flexibility is also present in its ability to support multiple distance measures inside a single index. The only aspect that changes is the different representation of the query (rapidly calculated at query time) for each distance measure. The index generation steps are shown in Figure 31.

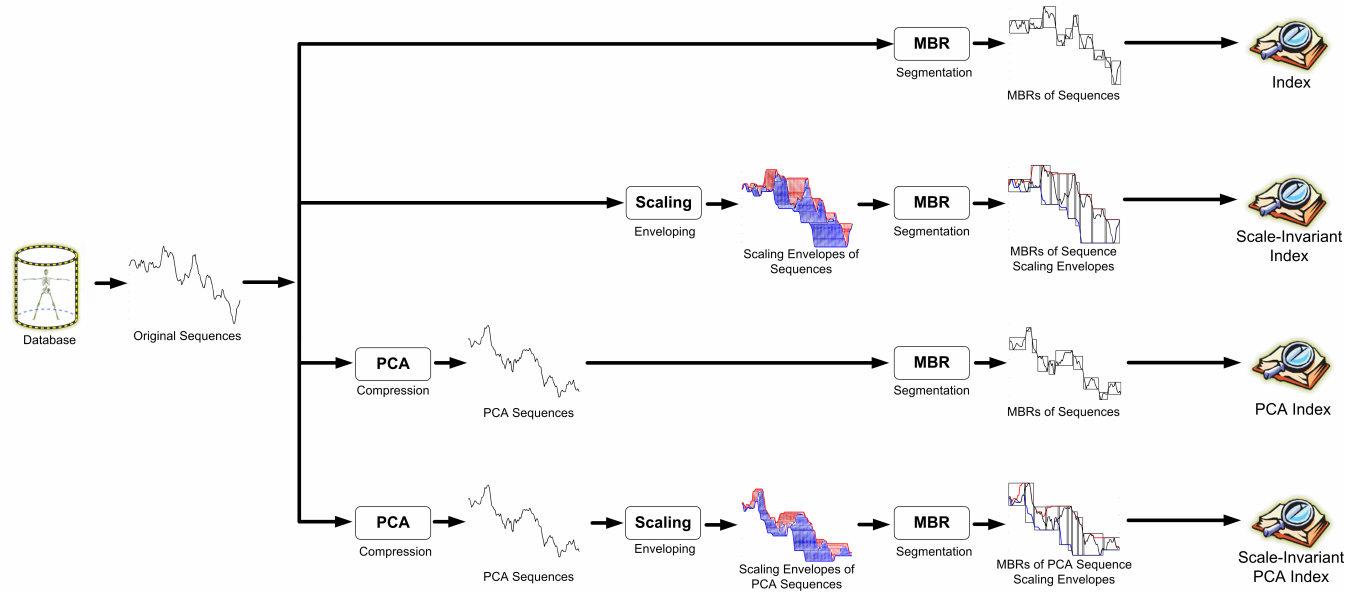

Figure 31: **Index Generation Steps**. An approximation for each mocap sequence in the database is stored in the index. (*Top fork*) When using the rotational-based body parameterisation, we obtain the approximation by the splitting of the joint angle curves into a series of Minimum Bounding Rectangles (MBR). (*Second fork*) If we want to support scaling-invariance within a pre-defined range, the angle curves are bounded by their scaling envelope which represents the original trajectory under all stretching conditions. Once again, the resulting envelope is then segmented into a series of MBRs. (*Third fork*) If we use the positional body parameterisation, which describes the body as a set of joint coordinates, the matching area is locked to whole body matching; but it guarantees meaningful and effective compressibility under Principal Component Analysis (PCA). This allows for faster whole body matching. (*Bottom fork*) The compressed PCA representation can also be made to support scale-invariance. Note that for clarity a one-dimensional curve is depicted; in practice, mocap data is multi-dimensional.

3.2.1.2 Matching Steps

We now describe the specifics of the matching process that are shown in Figure 32 in more detail. For a given single or multi-dimensional query, we construct a Minimum Bounding Envelope (MBE) that covers all the possible matching areas of the query under *local* warping conditions. This MBE is decomposed into MBRs. Keogh [2002] has shown that this eliminates false negatives. By allowing constrained time warping, we can achieve faster execution time and at the same time avoid distant and degenerate matches. The application of the Minimum Bounding Envelope only on the query suggests that user queries are not confined to a predefined and rigid matching length. The user can pose queries of variable warping in time. So, we can start by using a query with no bounding envelope, and increase it progressively in order to find more flexible matches.

Using the index we can discover which trajectories could potentially be similar to the query. We perform subsequence matching by shifting the query MBRs along all valid offsets of each database sequence. Based on the MBR distance (or overlap), similarity estimates are calculated between the query and the indexed subsequences and recorded at each instant. Note that the distance estimates are determined for each joint defined in the query and accumulated into a single global estimate (i.e. over all joint curves). These estimates 'guide' our search as to which sequences are most likely similar to the given query. For the most promising sequences, the exact distance function is calculated on the raw data, and the top-k results are kept in a priority queue.

The proposed framework can efficiently support more flexible matches by using the Longest Common Subsequence (LCSS) [Vlachos et al. 2002] or Dynamic Time Warping (DTW) [Keogh 2002] similarity models. As opposed to Euclidean distance, DTW and LCSS take into account variations in the time axis. The distance computed between the sequence MBRs is a lower-bound of the actual Euclidean or DTW distance. When LCSS is used, we use upper-bound MBRs since the LCSS model captures the similarity, which is inversely analogous to the distance. Therefore, according to the GEMINI framework [Agrawal et al. 1993], our index structure will guarantee no false negatives. The fundamental result of this observation is that our index will retrieve the same top-k matches as the ones that the sequential scan would have returned.

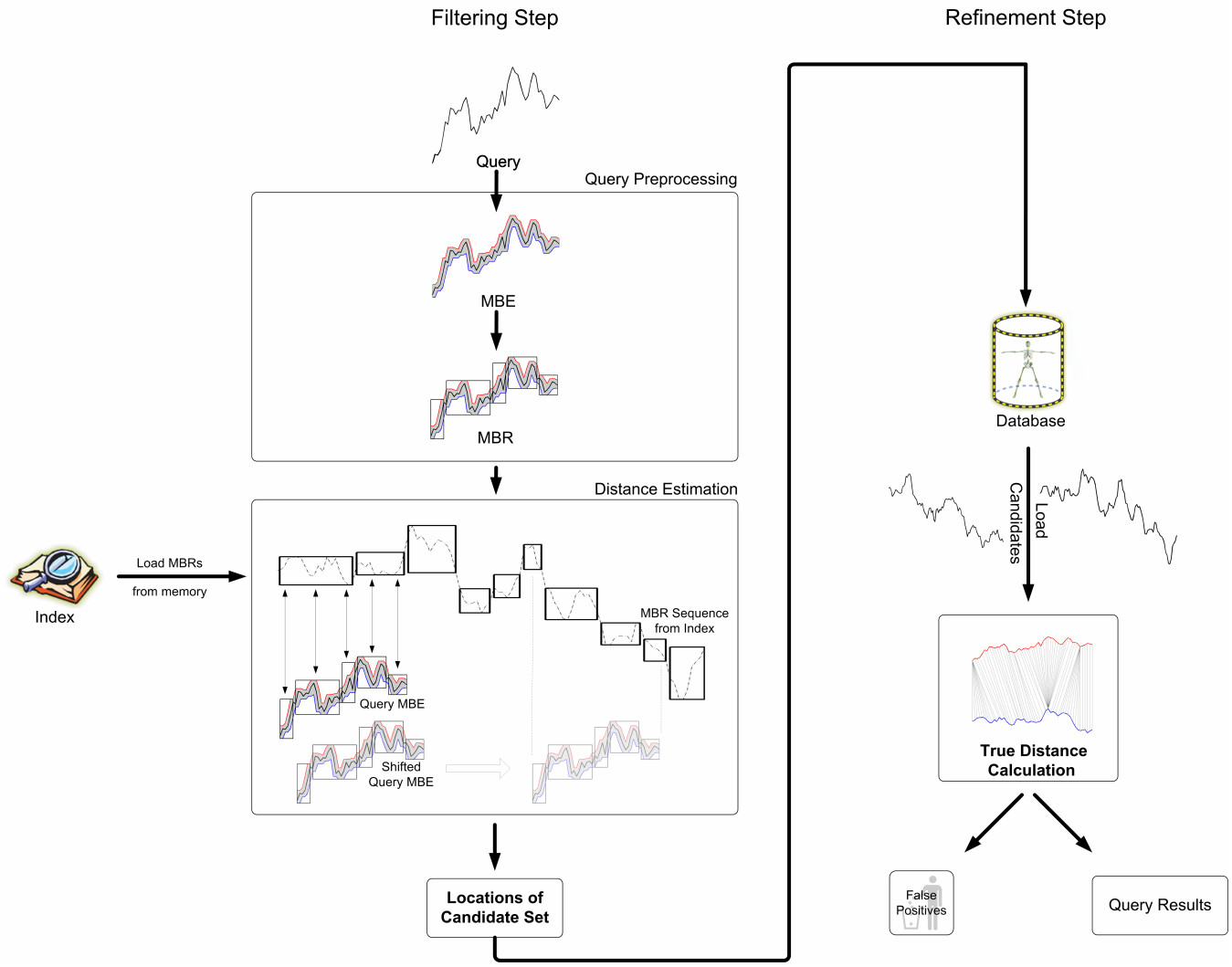

Figure 32: **Matching Process**. Given a query, we first enter the query preprocessing phase which occurs at query-time. The query is enclosed in a Minimum Bounding Envelope (MBE) that covers all the possible matching areas of the query under *local* warping conditions. This MBE is then decomposed into MBRs. Next, the index is used to discover which trajectories could potentially be similar to the query. We perform subsequence matching by shifting the query MBRs along all valid offsets of each database sequence. Based on the MBR distance (or overlap), similarity estimates are calculated between the query and the indexed subsequences and recorded at each instant. These estimates 'guide' our search as to which sequences are most likely similar to the given query. For the most promising sequences, the exact distance function is calculated on the raw data, and the top-*k* results are kept in a priority queue. Note that for clarity a one-dimensional curve is depicted; in practice, mocap data is multi-dimensional.

3.2.2 Motion Capture Data Representation

Before matching can begin, we need to define what we will be matching over. The format of the human motion comprising our mocap database is covered first. For matching purposes, we transform the original mocap data into a more manageable form in Section 3.2.2.2 . This involves defining the representation of each joint of the animated human body.

3.2.2.1 Motion Capture Database Format

Obtaining motion capture data can be difficult and expensive; but fortunately Stanford University's CS448 course [Stanford 2000] provides freely downloadable motion donated by House of Moves [House of Moves 2002]. This forms our database. It comprises 208 BVH files totalling 54 minutes of continuous motion, or 77865 frames at 24 frames-per-second (fps). The sequences are an eclectic set of short activities from generic motions such as standing, walking and running, to modern dancing, ballet and handling motions. The shortest sequence is about 3 seconds long (or 60 frames) and the longest about 54 seconds long (or 1303 frames). The skeleton consists of 17 joints, each described by 3 Euler angles. This gives 57 degrees of freedom including the root's spatial coordinates and the body's global orientation, both defined in world coordinates.

The database is deliberately left unlabelled and unsorted and consists primarily of a series of short sequences. Nonetheless, the matching and editing techniques presented in this chapter work equally well on one or more extended motion sequences. For example, an hour-long ballet performance or boxing match would both form perfectly viable candidates for our database.

3.2.2.2 Joint Parameterisation

The mathematical representations we use for poses and the motion making up an animation have an impact on the complexity, stability and performance of our matching techniques. For our purposes, the ideal joint primitive satisfies the following requirements:

> **Compactness.** A compact primitive represents a joint in the minimum required set of numbers. This reduces the dimensionality of the motion, consequently decreasing redundancy and increasing matching speed.
>
> **No singularities.** Singularities affect the robustness of the distance metric.
>
> **Smooth Interpolation**. The ability to interpolate smoothly between sequences of keyframes. During scale-independent matching, each candidate sequence is re-sampled to fit the user query ranges. To minimize query response time, the interpolation operation must be computationally efficient without introducing artefacts.

Fast and robust distance metric. The ability to meaningfully distinguish between joint configurations lies at the heart of the matching process. Singularities inherent in each primitive affect the distance estimates. The distance calculation is also the most frequently used operation during matching; computational efficiency is therefore essential.

Support for lower-bounding. The matching process uses a lower-bound of the true distance between poses to quickly eliminate potential candidate matches. Therefore we must be able to meaningfully enclose a series of rotations in a multi-dimensional minimum bounding box (MBB). The minimum possible distance between the query and candidate MBBs must be a lower-bound on the true rotational distance.

Unfortunately, as we will see, some of these requirements are mutually inconsistent. In the remainder of this section, we will evaluate how each of the standard joint parameterisations performs under the above requirements.

3.2.2.3 Rotational Body Representation

The most common way of representing a joint is to use rotations such as Euler angles, angle-axis and quaternions. The standard Euler representation does not allow smooth interpolation and robust distances to be directly calculated. Quaternions could be used but require costly constraints to be imposed to smoothly interpolate and compute distances. We therefore select the minimal angle-axis parameterization defined as $\boldsymbol{\omega} = \boldsymbol{a}\theta$ where $\boldsymbol{a} \in \Re^3$ is the axis of rotation and θ the rotation angle about $\boldsymbol{a}$ (Appendix A discusses this in more detail). Favoured in recent motion capture work [Bregler and Malik 1998; Molina-Tanco and Hilton 2000; Alexa 2002; Lee and Shin 2002], the angle-axis fits our initial requirements such as compactness and support for computationally efficient smooth interpolation and distance calculations. With angle-axis joints, the distance between two character poses P_1 and P_2 is defined as:

$$d(P_1, P_2) = \sum_{j=1}^{J} w_j \left\| \omega_1^j - \omega_2^j \right\|^2$$

J is the total number of joints in the character hierarchy.
w_j is a weighting value for each joint
$\omega_1^j \in \Re^3$ is the compact angle-axis vector at the j-th joint of pose P_1
$\omega_2^j \in \Re^3$ is the compact angle-axis vector at the j-th joint of pose P_2

Weights w_j are introduced to take into account the impact of each joint rotation on the overall skeletal hierarchy. Otherwise, minor joints such as the hand would have as much influence on the final pose-to-pose distance as a more significant thigh rotation of the same magnitude. It is clear that the visual effect of the rotation of the hand is not the same as the thigh rotation. Until recently, these weights were determined manually [Lee et al. 2002; Arikan and Forsyth 2002] before Wang and Bodenheimer [2003] established perceptually optimised weights by introducing user study data into a constrained least-squares technique.

3.2.2.4 Positional Body Representation

Instead of considering the character's pose as a set of rotations, we can consider the spatial coordinates at each joint. Forward kinematics is used to calculate the Cartesian coordinates at each joint given the joint angles of any parent bones in the body hierarchy. Each joint i provides a local rotation R_i and a local translation T_i. Concatenating R_i and T_i gives M_i^{local}, the local transform at each joint. The transformation of the point x on joint j is found by concatenating all previous transforms in the hierarchy as follows:

$$x' = M_x^{global} x = \prod_{i=0}^{j} M_i^{local} x = \prod_{i=0}^{j} T_i^{local} R_i^{local} x$$

where $i = 0$ is the bone at the root of the hierarchy (i.e. the hips) and x' is the world space position of joint j.

The spatial pose representation avoids many issues faced by its rotation-based counterparts. The distance between two positions is simply the Euclidean distance and it supports any Euclidean interpolant. No joint weighting is required in principle as the distances in the pose space are proportional to visual disparity [Molino-Tanco 2003]. This constitutes a fundamental advantage over the above rotation-based metrics when linear dimensionality reduction techniques, such as PCA, are applied. The PCA normalisation process nullifies the influence of weights assigned to each joint in the orientation-based pose parameterisation. This is because there is no prioritisation in the data so all joints are given the same importance in the compression. The spatial representation does not suffer from this difficulty (details in Section 3.2.7). Other advantages include its compactness (i.e. only 3 numbers) and support for lower-bounding.

The distance between two poses P_1 and P_2 is the result of the square of the distances between respective joint coordinates:

$$d(P_1, P_2) = \sum_{j=1}^{J} \left\| \boldsymbol{x}_1^j - \boldsymbol{x}_2^j \right\|^2$$

$\boldsymbol{x}_1^j \in \Re^3$ is the Cartesian coordinate at the *j*-th joint of pose P_1
$\boldsymbol{x}_2^j \in \Re^3$ is the Cartesian coordinate at the *j*-th joint of pose P_2

The disadvantage of this parameterisation is the dependence between each joint's Cartesian coordinates. We cannot select the matching area dynamically by including or excluding joints. For example, if we narrow the matching area to the hand and elbow joints, the resulting distance measures will be dominated by the configuration of the shoulder and spine joints. This means that matching is principally limited to whole-body matching.

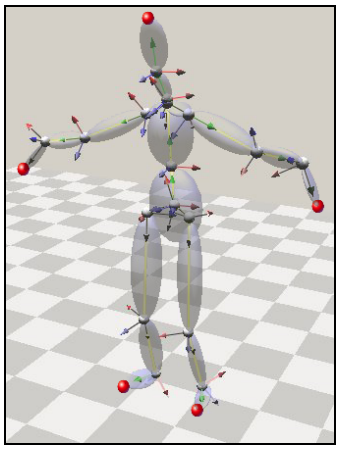

Figure 33: The five end-effectors for this character are depicted by the red spheres.

Another by-product is the increased dimensionality due to the introduction of five positional sites, known as *end-effectors*. End-effectors are illustrated in Figure 33. These correspond to the extremities at the hands, the feet and the scalp. If the end-effectors were omitted, the hand, feet and head joint rotations would not be represented in the Cartesian parameterisation. An additional 5*3=15 Cartesian dimensions add to the existing 17*3=51 positional dimensions, totalling 66 to represent the whole body. As we will see in Section 3.2.7, the apparent high dimensionality includes a lot of redundant information and is effectively compressed using PCA.

Some motion synthesis algorithms also use velocity [Arikan and Forsyth 2002; Kovar et al. 2002; Lee et al. 2002]. However, Wang and Bodenheimer [2003] suggest that velocity components are not significant in the inter-pose metric for a wide variety of motions, and they are not considered here.

3.2.2.5 Supporting Global Coordinate and Rotation Invariant Distance Metrics

World space translation and rotation have not be been considered in the inter-pose distances. We observe that motions are fundamentally unchanged by rotations about the vertical axis and translations in the horizontal floor plane [Gleicher 2001; Kovar et al. 2002; Lee et al. 2002]. For example, a walking motion looks the same whether it starts at the origin and heads east or starts ten metres to the side and heads southwest. It is generally recommended to ignore the horizontal movement described in x and z translation components, along with rotation around the vertical axis. Similar motions, irrespective of the global position and vertical orientation of the body, will then only be returned. The skeleton therefore always keeps the same vertical orientation with respect to the world reference frame. Vertical translations are commonly included to help differentiate between postures, like for instance a crouch and jump in which the knees rise to the chest. The final choice is left to the user's discretion since individual contributions of each global orientation and rotation component can be weighted at query time.

Supporting weights for global orientation and translation over the Cartesian coordinate-based parameterisation is problematic. Although its inter-pose distance function remains unchanged, the spatial data itself does not. The Cartesian coordinates for each pose are determined once using forward kinematics during the database pre-processing phase. This is a costly operation over a large database containing thousands of poses. This results in the initial weights assigned to the global transform being incorporated in each joint coordinate. Changes made to the global transform weights at query time can only take effect if the coordinates are recomputed for each candidate motion sequence. This entails calculating the concatenation of the inverse global transform with the new global transform at each joint and applying it to each coordinate throughout. Resizing the MBRs would also be necessary. Although feasible for global translational changes (since only a series of subtractions are necessary), global orientation changes at query time would be computationally prohibitive.

It makes more sense to use coordinates based on a vertically centred local skeleton frame, rather than with respect to a global coordinate frame. This avoids the distance metric being dominated by differences in

global location or orientation, rather than differences in arrangements of the skeleton and therefore visual disparity. For the same reasons as above, only the vertical global displacement is applied to the local frame.

Finally,
Table 1 compares the considered representations in terms of compactness, disposition to singularities affecting the inter-pose metrics, interpolability and computational efficiency.

	Euler	**Quaternion**	**Angle-axis**	**Positional**
Parameters per joint	3	4	3	3
Parameters for whole skeleton	3*17 joints + 3*global_rotation + 3*global_translation = 57	4*17 joints + 4*global_rotation + 3*global_translation = 75	3*17 joints + 3*global_rotation + 3*global_translation = 57	3*17 joints + 3*global_translation + 3*5 extremities = 69
Joint parameterisation redundancy	✘	✓	✘	✘
Domain distance function	Euclidean distance	Geodesic norm	Euclidean distance	Euclidean distance
Distance calculation speed	Fast	Slow	Fast	Fast
Weights required	✓	✓	✓	✘
Domain interpolation function	Linear interpolation	Spherical linear interpolation	Linear interpolation	Linear interpolation
Interpolation Quality	Bad	Exact	Good (on dense curves)	Exact
Interpolation speed	Fast	Slow	Fast	Fast
Candidate for linear dimensionality reduction	✘	✘	✘	✓

Table 1: **Comparison of Body Parameterisations**.

3.2.3 Calculating Motion Similarity

Having defined the joint representation and the manner in which it is indexed, we turn our attention to the issue of similarity measurement. In order to find relevant matches, we need a meaningful way to compare any two given motion sequences. Our similarity measure therefore takes two motion sequences as input and outputs a value indicating the strength of their mutual similarity. It should have the following desirable features:

- Mimic the human perception of similarity between any given two motions.
- Fast to compute.
- Configurable, so that the animator can tailor it to his or her perception of similarity.
- Possibility to lower-bound the true distance, so that we can quickly calculate an indicative and correct estimate of the true similarity to accelerate matching and allow indexing (see Section 3.2.4 for details).

After some notational preliminaries in Section 3.2.3.1 , we introduce and discuss three distance measures: the Euclidean distance in Section 3.2.3.2 , the DTW in Section 3.2.3.3 and the LCSS in Section 3.2.3.4 . Finally, we conclude on their effectiveness in Section 3.2.3.5 .

3.2.3.1 Data Representation Preliminaries

Under the Cartesian coordinate parameterisation, the matching area is fixed over the whole body corresponding to 69 separate matching dimensions. With the angle-axis parameterisation, the number of joints involved in the matching is dynamically set by the user at query time, along with inclusion of global position and rotation. Therefore the potential number of matching dimensions varies over the interval $D \in [1,57]$.

Let Q and C be two motion sequences composed of D dimensions spanning n and m keyframes respectively. Q and C correspond to two multi-variate time-series with elements defined as numeric values: $c_i, q_j \in \Re^D$, where D depends on the selected matching areas and body parameterisation. As a notational convention, when referring to the *i*-th element of a time-series of dimension *d*, we use an extra subscript e.g. $c_{d,i}$. We write Q and C as:

$$Q = ((q_{1,1}, q_{2,1}, \ldots, q_{d,1}, \ldots, q_{D,1}), \ldots, (q_{1,i}, q_{2,i}, \ldots, q_{d,i}, \ldots, q_{D,i}), \ldots, (q_{1,n}, q_{2,n}, \ldots, q_{d,n}, \ldots, q_{D,n})) \text{ and}$$

$$C = ((c_{1,1}, c_{2,1}, \ldots, c_{d,1}, \ldots, c_{D,1}), \ldots, (c_{1,j}, c_{2,j}, \ldots, c_{d,j}, \ldots, c_{D,j}), \ldots, (c_{1,m}, c_{2,m}, \ldots, c_{d,m}, \ldots, c_{D,m}))$$

For convenience, we summarize our notation in the following table:

C	Multi-dimensional candidate motion sequence
Q	Multi-dimensional query motion sequence
$i:j$	Indices i to j
$C[i]$	The i-th elements over all dimensions of C
$C[i:j]$	Subsequence of C containing elements from i to j
$c_{d,i}$	The i-th element of dimension d of candidate C
$q_{d,i}$	The i-th element of dimension d of query Q
$\|Q\|$	The number of keyframes in Q, also referred to as n
$\|C\|$	The number of keyframes in C, also referred to as m
D	The total number of dimensions to match over
$w_{i,d}$	The weight value for the i-th element of dimension d of query Q

Table 2: Notation used throughout this chapter.

3.2.3.2 Euclidean Distance

Most of the related work on multi-dimensional time-series has concentrated on the Euclidean distance. Its popularity is almost certainly due to its ease of implementation, its computational efficiency and that its metric property allows efficient indexing by dimensionality reduction techniques [Roddick and Hornsby 2000; Kim et al. 2001]. If we include weights w_d giving relative importance to different joints, the Euclidean distance (Figure 34) between two equal length sequences is:

$$d_{wEuclid}(Q,C) = \sqrt{\sum_{i=1}^{n}\sum_{d=1}^{D} w_d \left| q_{d,i} - c_{d,i} \right|^2} \quad \text{with} \quad w_d \geq 0$$

The animator can therefore assign more importance to certain joints in the distance metric by increasing their respective weights. Take the example where the upper body DoFs have large weights and the lower body DoFs small ones. Valid matches are the ones that exhibit very similar upper-body motion and only slightly similar lower-body motion. Since the square root function is monotonic and concave, we can remove the square root step and use the squared Euclidean distance:

$$d_{wEuclid}(Q,C) = \sum_{i=1}^{n}\sum_{d=1}^{D} w_d \left| q_{d,i} - c_{d,i} \right|^2 \quad \text{with} \quad w_d \in \Re$$

This gives identical rankings with the added bonus of being slightly faster to compute [Keogh and Kasetty 2002]. The principal motivation is that we can now introduce negative weights. In other words, body matching areas that are negatively weighted will lower the ranking of potential matches with similar motions to the query (for that same matching area). For example, we might want to find all motions with similar leg motions, but different arm motions.

We extend the above formulation to support time-varying multi-dimensional weights. In other words, we assign a multi-dimensional weight at every key-frame in the sequence, and the new distance therefore becomes:

$$d_{TwEuclid}(Q,C) = \sum_{i=1}^{n}\sum_{d=1}^{D} w_{d,i} \left| q_{d,i} - c_{d,i} \right|^2$$

At any keyframe in the query, the animator can assign more importance to certain joints in the distance metric by increasing their weight. For example, we might want to find a waving motion in the middle of a walking motion. Therefore, we would give more weighting to the upper body DoFs in the middle of the query only to make sure matches with similar waving motions in walking sequences are returned. This works well over the linear scan matching method. Unfortunately, it makes it very difficult to index with conventional indexing methods [Keogh and Pazzani 2000]. In Section 3.2.3.4 , we provide an indexable alternative based on the LCSS distance with functionally similar behaviour.

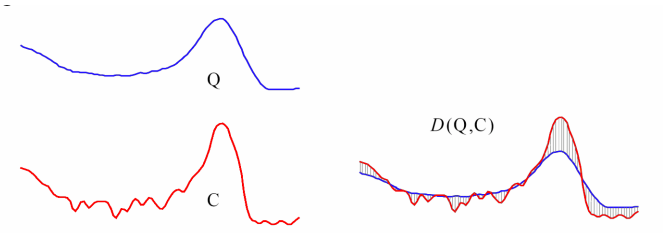

Figure 34: **The intuition behind the Euclidean distance over a single joint**. The Euclidean distance can be visualized as the square root of the sum of the squared lengths of the vertical lines in grey.

Although such a distance can be calculated efficiently, the drawback of the Euclidean metric is the tendency of outliers to dominate the distance measure. It is also very sensitive to small variations in the time axis [Agrawal et al. 1995; Yi et al. 1998; Chu et al. 2002] even though two motion sequences could be very

similar, but not perfectly synchronized. It also does not perform well in the presence of noise which is frequently introduced by motion capture systems.

3.2.3.3 Dynamic Time Warping

The problem of distortion in the time axis faced by the Euclidean distance is addressed by Dynamic Time Warping (DTW) [Kruskall and Liberman 1983; Rabiner and Juang 1993]. Originating from speech processing [Waibel and Lee 1990], and used in bioinformatics for gene expression data alignment [Clote et al. 2003] and motion capture editing [Bruderlin and Williams 1995; Kovar and Gleicher 2003], Bernt and Clifford [1994] were the first to use DTW for data mining (Figure 35).

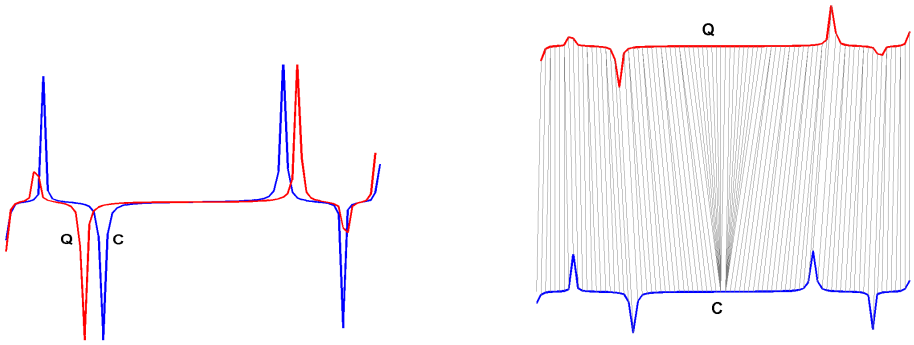

Figure 35: **Dynamic Time Warping**. (*Left*) Sequences Q and C are similar one-dimensional sequences that are locally out of phase. (*Right*) The corresponding indices alignment between Q and C obtained using DTW.

Given that $Head(Q) = ((q_{1,1}, q_{2,1}, \ldots, q_{D,1}), \ldots, (q_{1,n-1}, q_{2,n-1}, \ldots, q_{D,n-1}))$ corresponding to all the elements of Q except the last one, then the recursive definition of the Dynamic Time Warping (DTW) distance between two sequences Q and C of any length is:

$$\begin{aligned} DTW(Q,C) = d(Q,C) + \\ \min\{&DTW(Head(Q), Head(C)), \\ &DTW(Head(Q), C), \\ &DTW(Q, Head(C))\} \end{aligned}$$

where d is a distance measure allowing the similarity assessment of keyframes between Q and C. We call this the local distance measure as opposed to the global distance returned by DTW. We extend the definition of the DTW to weighted DTW (wDTW) to deal with weighted multi-variate time-series. This is

achieved by using the previously introduced weighted Euclidean distance $d_{wEuclid}$ as our local distance function.

Let $\mathrm{A} \in \Re^{n \times m}$ be a matrix where $\mathrm{A}_{ij} = d_{wEuclid}(q_i, c_j)$ over all D dimensions of Q and C. Then the goal of DTW is to find the path P that minimizes the cumulative sum of local distances $d_{wEuclid}$ along matrix A (i.e. from the cumulative sum from the bottom left to the top right of A). A path $P = p_1, p_2, \ldots, p_k, \ldots, p_K$ is defined as a series of indices into matrix A with indices $p_k = (i, j)_k$. The length K of all paths P satisfies:

$$\max(m, n) \leq K \leq m + n - 1.$$

The path P that wDTW recovers is the one with minimum accumulated cost, i.e.

$$wDTW(Q, C) = \min_P \left\{ \frac{1}{K} d_{wEuclid}(p_k) \right\}$$

where $d_{wEuclid}(p_k) = d_{wEuclid}(q_i, c_j) = \mathrm{A}_{ij}$. K is a function of P since warping paths can vary in length. We divide by K to compensate for paths of unequal lengths.

The solution to this minimisation problem is accomplished by dynamic programming (DP). DP finds the least cost path in a grid γ of size (*n*,*m*). It works by first evaluating the least cost distance to reach every square in the grid and then tracing back the path which corresponds to the overall least cost distance. By relying on the observation that there are three ways to get to any grid square (horizontally, vertically or diagonally), it accumulates the pre-calculated local distance $A_{ij} = d_{wEuclid}(i, j)$ per grid square $\gamma(i, j)$ with the minimum of the cumulative distances of the adjacent squares:

$$\gamma(i, j) = d_{wEuclid}(i, j) + \min\{\gamma(i-1, j-1), \gamma(i-1, j), \gamma(i, j-1)\}$$

Several constraints are exercised on the optimal path P since the number of possible warping paths grows exponentially with the length of the time-series. These constraints typically include boundary conditions (diagonally opposite start and finish squares), continuity (restriction to adjacent squares) and monotonicity (continuously spaced squares in time). The number of potential paths can be further reduced by applying the windowing condition [Berndt and Clifford 1994]. The allowable squares of matrix A are restricted to those that fall into a warping window $\left| i - \left(\frac{n}{m/j} \right) \right| < \delta$, where $\delta \in \Re^+$ is the window width. This is because

a meaningful alignment path is unlikely to drift very far from the main diagonal otherwise degenerate point correspondence between curves could occur. The distance that the path is allowed to roam is limited to the warping window (or *band*), directly above and to the right of the diagonal. We will refer to the warping window as the *temporal threshold*.

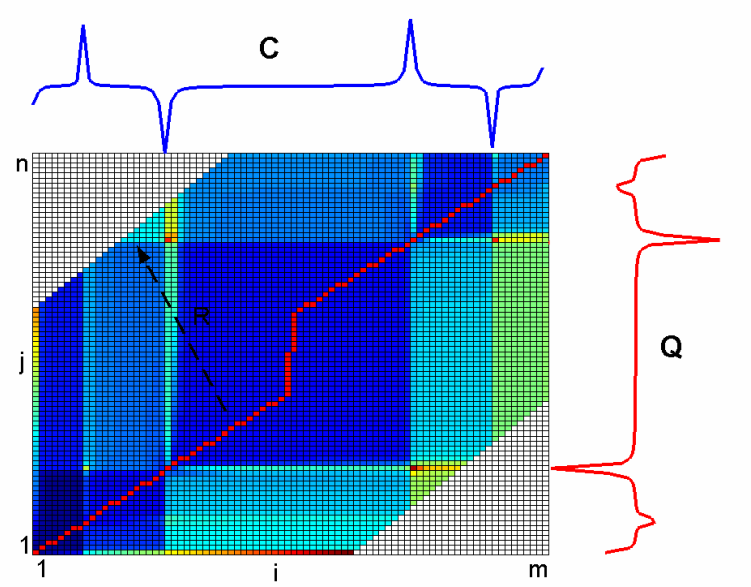

Figure 36: **DTW Grid and Path Visualisation**. The colour intensities on the search grid γ represent the accumulated distance at each position along sequences Q and C from Figure 35. Grid γ is used to find the optimal warping path (*red*). The local distance calculations and path search are limited to be within the warping window (i.e. temporal threshold) of length $\delta = 50$ with $|Q| = 100$ and $|C| = 105$.

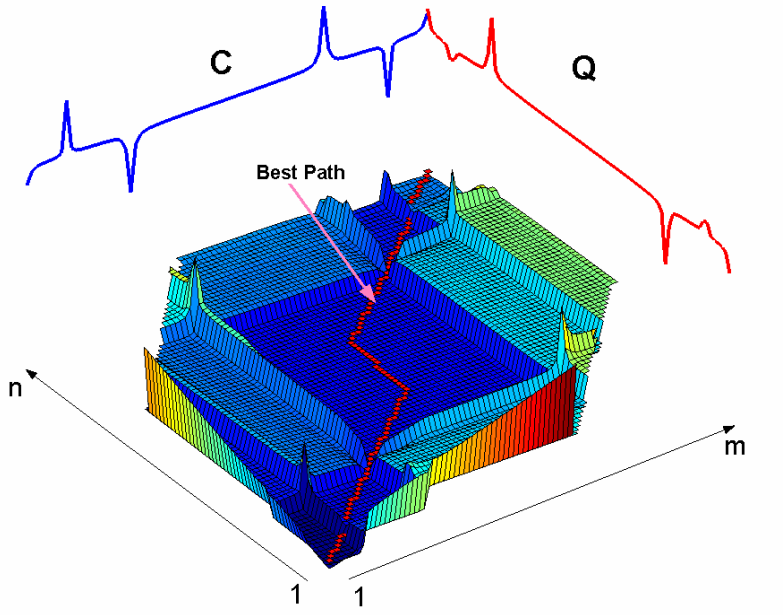

Figure 37: **DTW Grid and Path 3D Visualisation**. This is a height-field rendition of the distance accumulation grid γ along with the optimal path in red. The distance magnitude increases with height.

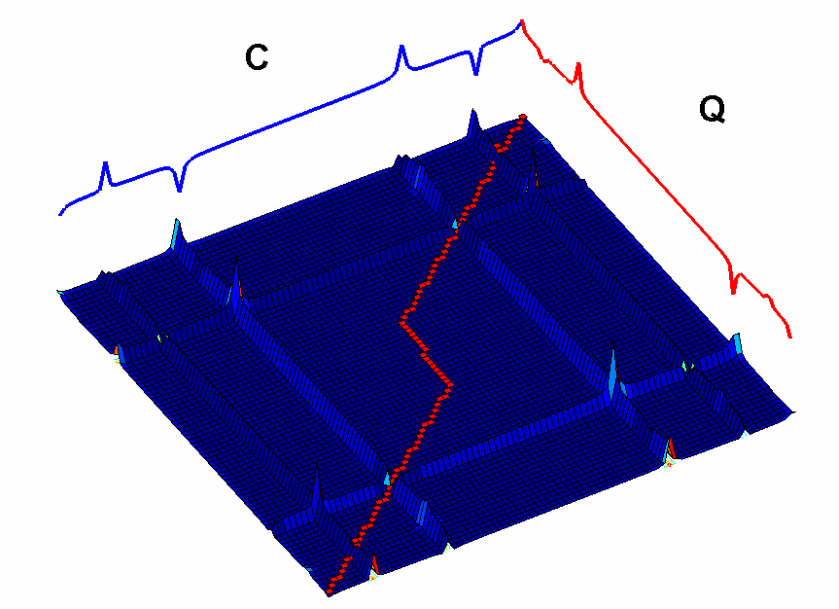

Figure 38: **Point-to-point Distances**. This height-field represents the Euclidean distance from every position in *Q* to every other in *C*.

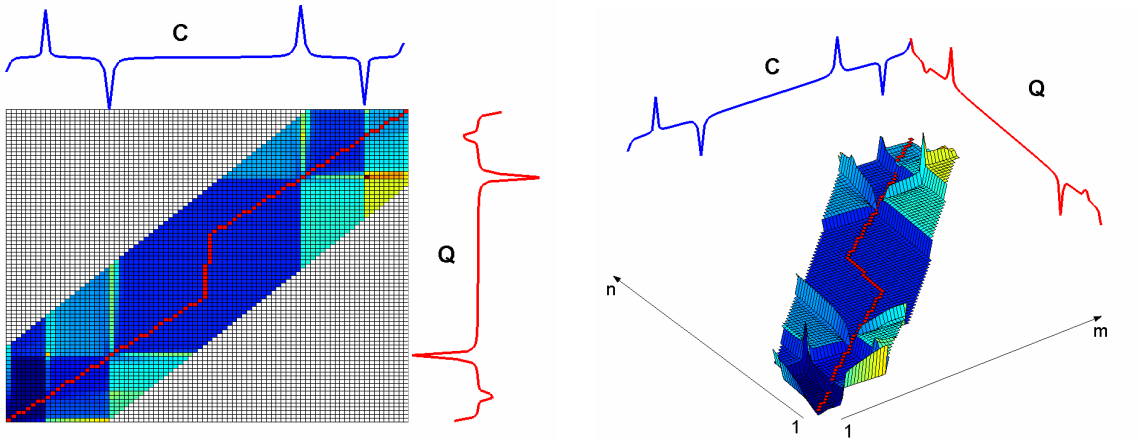

Figure 39: **Smaller Warping Window**. Same as Figure 36 with a warping window decreased from $\delta = 50$ to $\delta = 20$, considerably reducing the search space. The final distance and path remains unchanged whilst the amount of computation is decreased substantially.

The Euclidean distance has difficulty dealing with sequences of different length. A way round this is to pad or resample one of the sequences to meet the other's length. The wDTW measure supports matching between sequences of different length. If both sequences are the same length and the warping window is null then wDTW is equivalent to the weighted Euclidean distance. Many researchers [Tappert and Das 1978; Rabiner 1993; Keogh 2002] use a constant warping window over the whole length of the longest

sequence. Its width is typically set at around 10% of the longest sequence [Sakoe and Chiba 1978]. This is because in most datasets there is no need to perform full length warping as the global DTW distances converge after a certain warping window. Allowing for larger windows would not yield changes in the similarity at the expense of prolonged execution time. For example, the final DTW distance in Figure 36 (also in Figure 37 and Figure 38), with a window $\delta = 50$, is the same as in Figure 39 for a reduced window of $\delta = 20$. In addition to helping to speed up DTW pruning off many costly distance and path computations, the warping window prevents a pathological warping which might affect distance accuracy [Keogh 2002]. The reason is that a relatively small section of one sequence is allowed to map onto a relatively large section of another. This is easily explained since excessive matching envelopes can force different movements in the query to match each other.

Previous work considered variations of a limited set of fixed shapes of the warping window [Itakura 1975; Sakoe and Chiba 1978]. We introduce the notion of a user-defined time-varying warping window $\delta \in \Re^n$ with a value for every position along the length *n* of the query sequence. This allows the animator to regulate the temporal leniency throughout the query, which is why we refer to it as the temporal threshold. Figure 40 shows the DTW search grid with a linearly increasing temporal threshold. The exact speed and timing of certain gestures in the query can be highly constrained whilst other portions can be left more relaxed. For example, given a query containing a punch followed by a kick, we might only want matches with the exact same punch speed occurring at the same instant followed by a fast or slow kick with similar spatial amplitude some time afterwards. The animator achieves this over the query by narrowing the temporal threshold during the punch while enlarging it during the kick.

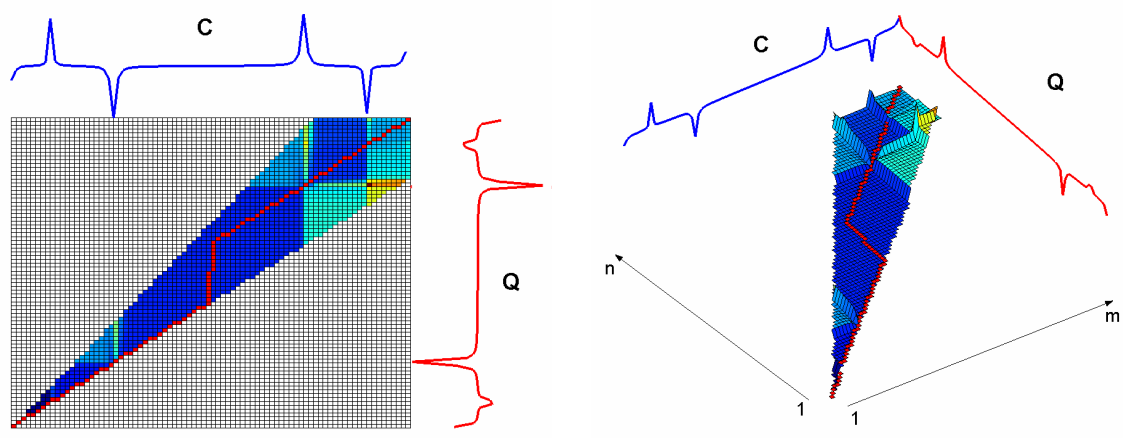

Figure 40: **Time-varying Warping Window**. Same as Figure 36 except that the temporal threshold (i.e. warping window) linearly increases in size from $\delta_{start} = 0$ to $\delta_{end} = 20$.

The current definition of wDTW optimises temporal warping over *all* dimensions simultaneously. In other words, the joint timings are warped in unison. This makes sense since there is an inherent correlation between body parts [Shimada and Uehara 2003]. The synchronisation of the original movement would be distorted if warping was applied to each joint separately. For example, a human walk is characterised as an ordered sequence of movements described by the swing of one arm followed by that of the opposite leg. Therefore, the temporal threshold at each keyframe is the same over all dimensions. Joints (i.e. dimensions) with larger user-weights will have more influence on the calculated warping path, and consequently on the final distance measure.

The flexibility provided by DTW is offset by a number of factors. The most limiting is the lack of scalability over large databases because of its quadratic time complexity [Berndt and Clifford 1994; Keogh and Pazzani 2000b; Vlachos et al. 2003]. Given a constant temporal threshold δ, the time and space complexity required to compute DTW is $O(\delta(n+m))$. Depending on the length of the sequences, DTW is typically hundreds or thousands of times slower than the corresponding Euclidean distance. DTW is also not stable over noisy data since by matching all the points, it also matches the outliers distorting the true distance between the sequences. As the number of outliers increases, the accuracy of DTW deteriorates.

3.2.3.4 Longest Common Subsequence (LCSS)

Supporting time axis variations is an important feature of DTW. To avoid distant and degenerate matches under DTW due to noise, we use the Longest Common Subsequence (LCSS) model. It also supports time-shifting and unequal length sequence matching while ignoring noisy parts. With LCSS, motions that are close in space at different time instants can be matched if the time instants are also close.

LCSS is a variation of the *edit distance* [Bozkaya et al. 1997] specially adapted for continuous values. The edit distance between two alphanumeric sequences is the smallest number of insertions, deletions and substitutions required to change one string into another. Instead of checking equality between two elements, LCSS checks if the elements are within a *matching distance* of each other. In other words, it checks whether the relation $|q_i - c_i| < \varepsilon$ holds, where q_i and c_i are *i*-th elements of one-dimensional sequences Q and C respectively. The real number ε is referred to as the spatial matching threshold. In a similar manner to DTW, adaptable temporal tolerance is allowed. The temporal threshold δ controls how far in time a match in one sequence to a point can be from the equivalent point in another sequence. Two sequences are

matched by allowing them to stretch, preserving order but allowing some elements to be *unmatched*. This provides a more satisfactory measure of similarity between time-series by giving more weight to the similar portions of the sequences [Vlachos et al. 2003]. For example, when comparing any kind of data (e.g. images or motion), we mostly focus on the portions that are similar and where we are willing to pay less attention to regions of great dissimilarity.

The standard recursive definition of LCSS [Bozkaya et al. 1997; Vlachos et al. 2003] over two one-dimensional time-series Q and C is given by:

$$LCSS_{\delta,\varepsilon}(Q,C) = \begin{cases} 0 & \text{if } Q \text{ or } C \text{ is empty} \\ 1 + LCSS_{\delta,\varepsilon}(Head(Q), Head(C)) & \text{if } |q_n - c_m| < \varepsilon \text{ and } |n - m| \le \delta \\ max(LCSS_{\delta,\varepsilon}(Head(Q), C), LCSS_{\delta,\varepsilon}(Q, Head(C))) & \text{otherwise} \end{cases}$$

where $\delta \in \Re$ is the temporal threshold and $\varepsilon \in \Re$ is the spatial threshold (Figure 41). LCSS is efficiently solved using the same Dynamic Programming approach used for DTW. Again, like DTW, the complexity is in the order of $O(\delta(n+m))$ with a constant matching window δ in time [Das and Gunopulos 1997]

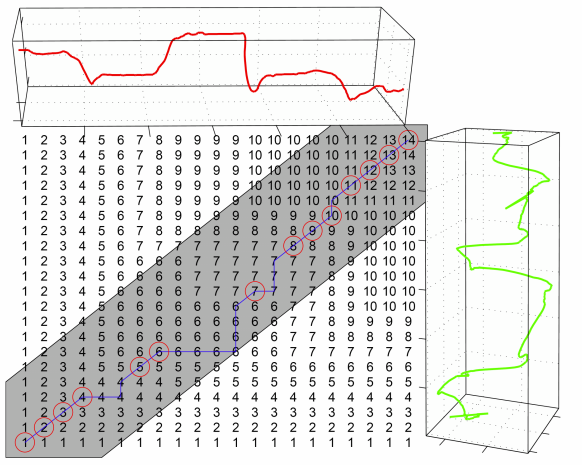

Figure 41. **Solution of the LCSS problem for a 2D motion using dynamic programming**. The grey area indicates the elements that are examined if we confine our search window to $\delta = 6$.

Like the Euclidean and DTW distances, the final value of LCSS is unbounded and depends on the length of the compared sequences. We can normalize it so that we can directly compare sequences of different lengths. Therefore the final distance measure $D_{\delta,\varepsilon}$ is defined as:

$$D_{\delta,\varepsilon}(Q,C) = 1 - S_{\delta,\varepsilon}(Q,C) \qquad \text{with} \quad S_{\delta,\varepsilon}(Q,C) = \frac{LCSS_{\delta,\varepsilon}(Q,C)}{\min(n,m)}$$

where $S_{\delta,\varepsilon}(Q,C)$ is the similarity function between sequences Q and C. We extend the original LCSS formulation to support multi-dimensional time-series with a user-defined time-varying temporal threshold $\delta \in \Re^n$ and spatial threshold $\boldsymbol{\varepsilon} \in \Re^{n \times D}$ such that:

$$LCSS_{\delta,\varepsilon}(Q,C) = \begin{cases} 0 & \text{if } Q \text{ or } C \text{ is empty} \\ 1 + \mathrm{LCSS}_{\delta,\varepsilon}(Head(Q), Head(C)) & \text{if } \forall d \in D,\ |q_{d,n} - c_{d,m}| < \varepsilon_{d,n} \quad \text{if } \varepsilon_{d,n} > 0 \\ & \qquad\qquad |q_{d,n} - c_{d,m}| > |\varepsilon_{d,n}| \quad \text{if } \varepsilon_{d,n} < 0 \\ & \text{and } |n - m| \le \delta_n \\ \max(LCSS_{\delta,\varepsilon}(Head(Q), C), & \text{otherwise} \\ LCSS_{\delta,\varepsilon}(Q, Head(C))) & \end{cases}$$

The time-varying temporal threshold δ behaves in much the same way as introduced for DTW. It enables the user to specify how far in time the algorithm can search for a match within the current spatial threshold.

The new formulation of the time-varying spatial threshold ε allows for powerful matching possibilities. For every potential match, the animator can specify by how much each of its joints can deviate at each point from that of the query motion. Thus, not only the exact speed and timing but also the spatial range, of certain gestures in the query can be highly constrained whilst other portions can be left more relaxed (Figure 42).

If the spatial threshold is positive (i.e. $\varepsilon > 0$), we refer to is as an *inclusion* threshold since only candidate matches with values *within* the range (i.e. $|q_{d,i} - c_{d,j}| < \varepsilon_{d,i}$) contribute to the final distance estimate. If the spatial threshold is negative (i.e. $\varepsilon < 0$), it becomes an *exclusion* threshold since only candidate matches with values *outside* the range (i.e. $|q_{d,i} - c_{d,j}| > |\varepsilon_{d,i}|$) contribute. This allows what we refer to as *inverse matching*, where sections of the query with negative spatial thresholds will be considered more similar if the candidate is outside the range. Therefore the greater the magnitude of $|\varepsilon_{d,i}|$, the greater the difference between the query motion and the best returned match (Figure 43). For example, given a query containing a punch

followed by a kick, we might only want matches with any type of movement but a punch followed by the exact same kick and another unspecified kick. The animator achieves this by adjusting the query's spatial threshold to a negative value during the punch followed by a small positive value during the first kick and a larger throughout the second kick. The determination of ε is application-dependent. In our experiments we use temporal thresholds similar to that of wDTW, i.e. 10 to 20 % of the query length [Sakoe and Chiba 1978]. The spatial threshold is set independently for each dimension and its value is equal to some proportion (usually 10% to 30%) of the standard deviation of the query on the corresponding dimension. This is to make the spatial threshold accommodate the different spatial positional or rotational ranges of each joint. For example, we would not want the neck's small spatial threshold to be applied to the highly variable shoulder joint. Otherwise, the matching zone for the shoulder joint would be too restrictive.

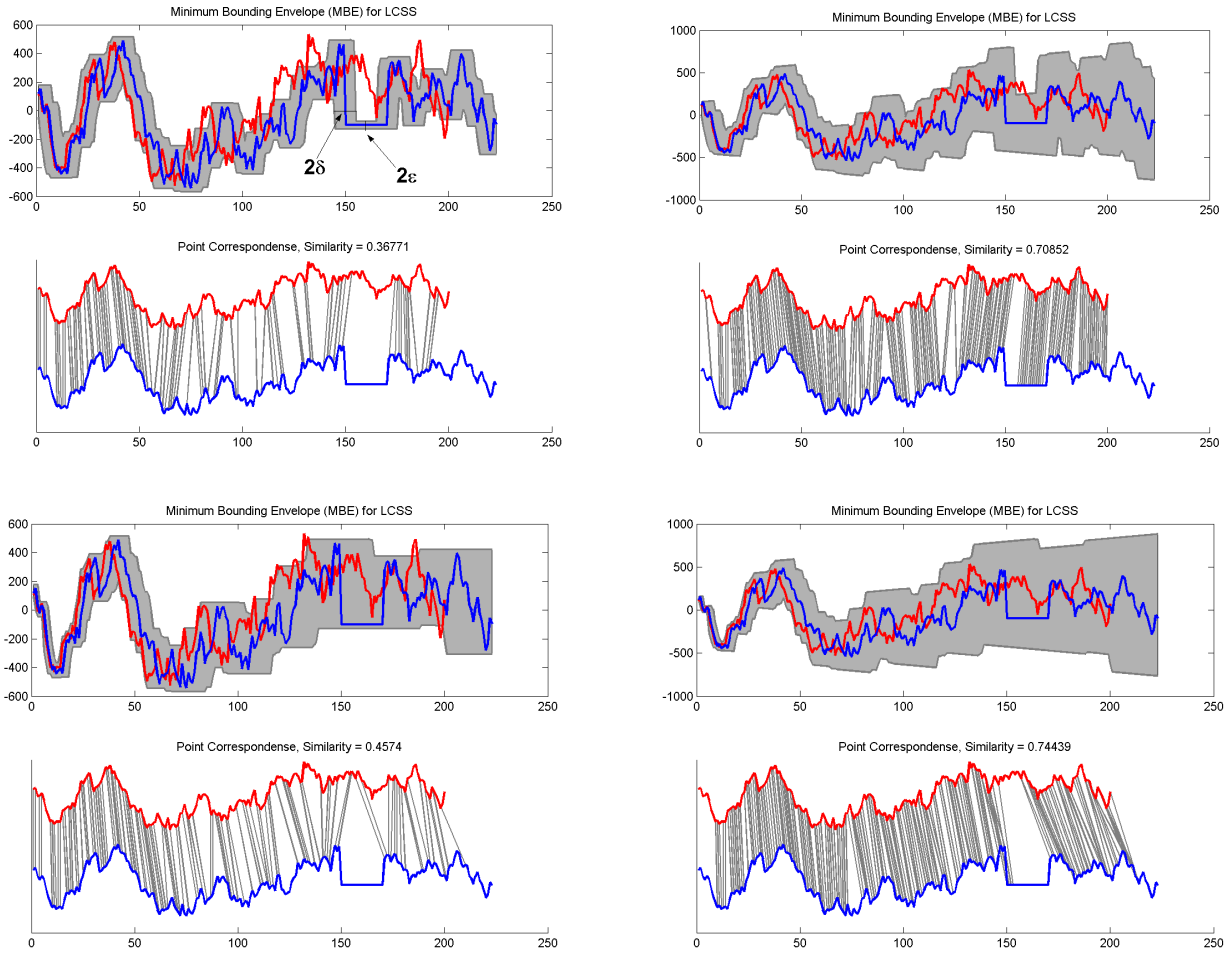

Figure 42: **Varying the LCSS thresholds over one-dimensional sequences**. The query sequence is in blue and the candidate sequence in red. (*Top-left*) LCSS matching is within the region delimited by constant thresholds δ and ε. Points in the two sequences within the grey regions can be matched by the LCSS function. The bottom graph shows the valid correspondences between both sequences. (*Top-right*) The spatial threshold ε is linearly increased throughout the query. (*Bottom-left*) Here we similarly increase the temporal threshold. (*Bottom-right*) Both the spatial and temporal thresholds are increased linearly. Notice the increase of the similarity measure along with the increased valid point correspondences.

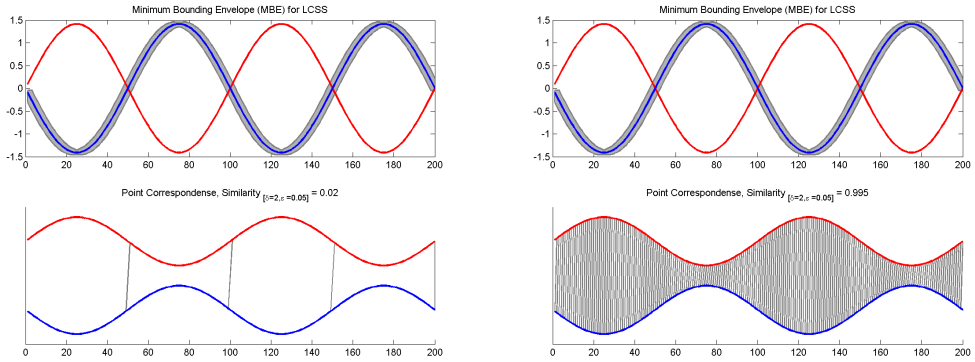

Figure 43: **LCSS inclusion and exclusion spatial threshold over one-dimensional sequences**. The query sequence is in blue and the candidate sequence in red. (*Left*) The query and the candidate consist of the same sinusoid curve shifted or not by π. With a small positive spatial threshold, the point correspondence and resulting similarity is low. This is to be expected since the motions are perfectly opposed. (*Right*) By negating the spatial threshold, inverse matching takes place. The *exclusion spatial* threshold only allows candidate matches *outside* the grey band to contribute to the final distance which, in this case, is most of the candidate.

The spatial thresholds have a similar function to the weights assigned to each dimension in $d_{wEuclid}$ and wDTW. The advantage here is that they are time-varying meaning that their values can change throughout the query. This achieves what we gained from the $d_{TwEuclid}$ distance function but with the added feature that it can be indexed efficiently (see Section 3.2.3.4). Decreasing the spatial threshold in LCSS is somewhat equivalent to increasing the weight in $d_{TwEuclid}$. The difference is that with $d_{TwEuclid}$, we are not sure to what extent the final distance value is influenced by our weights. With LCSS, we can guarantee that a match with similarity $S_{\delta,\varepsilon} = 1$ is completely within a range ε of the query (the opposite is true for inverse matching). Conditions like these are not possible with $d_{TwEuclid}$.

Note that the spatial threshold $\varepsilon = \infty$ is conveniently equivalent to the null weight in wDTW. Tagging a section of the query with the null weight is equivalent to stating that we do not care what happens in that region (Figure 44). For example, we might be interested in finding a motion with a kick, followed by three seconds of undefined motion leading to a final punch. The three second middle portion of the query would then be given an infinite spatial threshold, or equivalently, a null weighting.

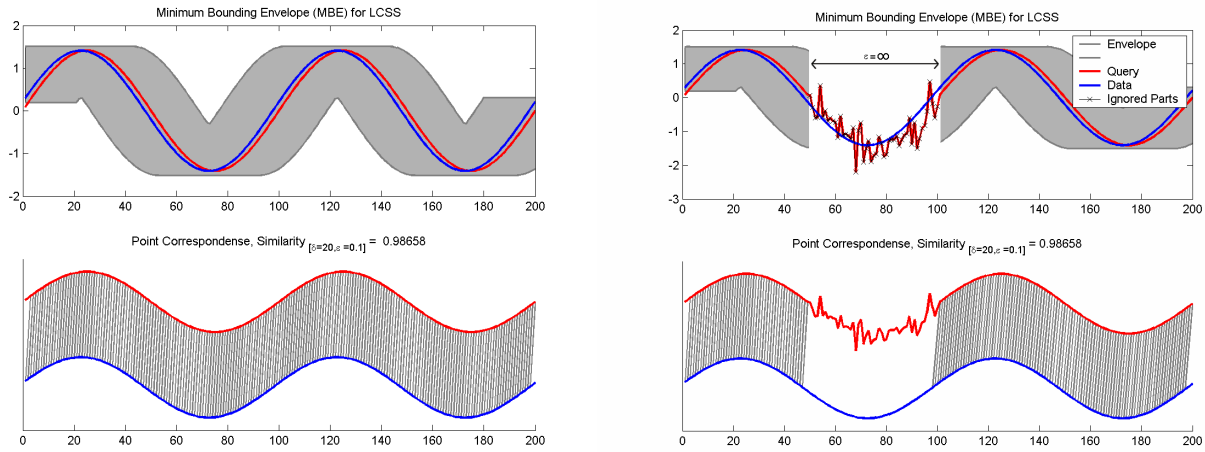

Figure 44: **LCSS *don't-care* spatial threshold over one-dimensional sequences**. The query sequence is in Red and the candidate sequence in Blue. (*Left*) The query and the candidate are two slightly shifted sinusoid curves and both fit inside the current spatial and temporal envelopes. Each position along the query finds a corresponding valid match in the candidate sequence Their similarity is therefore close to one. (*Right*) Random noise is added to the middle of the query. By giving the whole noisy section a spatial threshold $\varepsilon = \infty$, it can be ignored in the LCSS calculation. The correspondence diagram shows that the LCSS algorithm does not try to match up points during the *don't-care* tagged noisy region. As expected, the final LCSS measure is the same.

3.2.3.5 Conclusion

Three distance measures have been discussed: the weighted Euclidean distance, the weighted Dynamic Time Warp and the LCSS. Both the weighted Dynamic Time Warp and the LCSS measures represent a significant improvement compared to the Euclidean distance. However, they come at a substantially increased cost in computation. The Euclidean distance can quickly find close matches, while using LCSS and wDTW can distinguish more flexible similarities. Given the nature of mocap data, we argue that the use of LCSS and wDTW is preferable, since we want to match characters that move in a similar way, but at slightly different speeds. Figure 45 gives an indication of the differences.

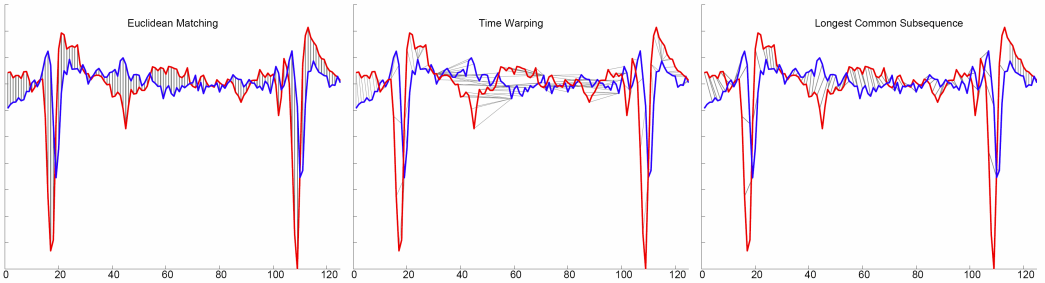

Figure 45: **Point correspondences between two one-dimensional curves**. Notice how the matching is rigid with the Euclidean distance, degenerates with DTW and is more stable, yet flexible with LCSS.

Although computationally equivalent to wDTW, LCSS has the advantage for two reasons. It effectively ignores the noisy sections of the sequences, frequently introduced by motion capture systems. It is also more configurable since an inclusion or exclusion spatial matching range can be set over each joint at *any position*. This guarantees that only values within certain ranges contribute to the final distance measure. Naturally, there is a cost associated with this. If the spatial thresholds are too restrictive and no motion within the database satisfies the matching range, LCSS will rate all candidates to a null similarity value. Practically, this means that no matches will be returned to the animator if the matching parameters are too constraining. With wDTW, we are always guaranteed a best match. Additionally, wDTW alleviates the need for the user to enter any matching parameters such as spatial threshold and even the temporal threshold. Therefore, the choice of the metric depends on user preferences. If it is essential that matches are returned in every case or no thresholds are specified, then wDTW is preferable. If it is essential that only matches satisfying strict parameters are returned, then LCSS is preferable.

The computational costs of wDTW and LCSS are many orders of magnitude higher than that of the Euclidean distance. It is therefore essential to limit the number of times these distance functions are executed. Our indexing scheme addresses this problem by pruning off unnecessary distance calculations. Table 3 summaries the merits and pitfalls of each distance measure.

	Euclidean	**DTW**	**LCSS**
Speed	Fast	Slow	Slow
Complexity (with sequences of length n and m)	$O(n)$ (with $n=m$)	$O(\delta(n+m))$ (with a constant matching window δ)	$O(\delta(n+m))$ (with a constant matching window δ)
Weights supported	✓	✓	✘
Temporal tolerance	✘	✓	✓
Spatial tolerance	✘	✘	✓
Requires user to specify tolerances	✘	✘	✓
Support for sequences of different lengths	✘ (not without preprocessing)	✓	✓
Always returns a match	✓	✓	✘
Sensitivity to noise	High	High	Low

Table 3: Distance function comparison table.

3.2.4 Estimating Motion Similarity

In the previous section, we defined three similarity measures. They are costly in I/O and computational resources. Therefore, a fast matching procedure can only be devised if we can quickly estimate a bound on the true distances in the index. This will help avoid unnecessary checks of the entire database for potential matches. This section details how the estimates are obtained. This leads us to examine the core of the matching algorithm in detail.

First, we present the estimation process for the wDTW distance. Since the Euclidean distance is a special case of wDTW, we do not consider it further. The estimation procedure is then integrated into our MBR index and extended to subsequence matching. We do the same for the LCSS distance.

3.2.4.1 Pruning wDTW Calculations

Similarity estimation under wDTW is very demanding in terms of CPU time. One way to address this problem is to use a fast lower-bounding function to help prune sequences that could not possibly be the best match. The bound must approximate the true wDTW distance and be as large as possible, since zero is always a valid, yet trivial, lower-bound.

This section formalises the chosen lower-bounding technique along with its extension to MBR sequences. Figure 46 summarizes the steps required to estimate the wDTW distance between a query Q and a candidate C from the database. Note how the approach also provides a lower-bound to the true Euclidean distance. It is simply a special case of wDTW with the temporal threshold set to a constant null. The speedup is not as dramatic since the execution efficiency of the Euclidean metric is mostly I/O bound, not CPU bound.

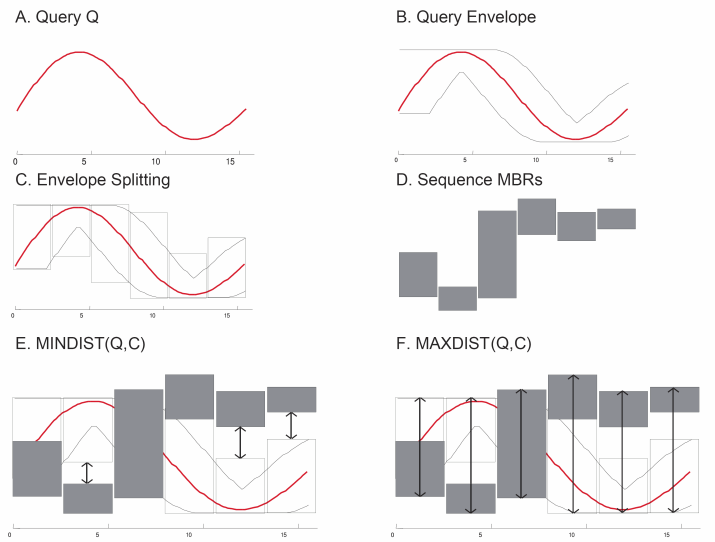

Figure 46: **An illustration of the necessary steps to estimate the wDTW metric between query *Q* and a candidate *C*.** One-dimensional sequences are chosen for clarity. The original user query (*A*) is first enclosed in a Minimum Bounding Envelope (MBE) (*B*) which delineates the possible matching areas given wDTW's temporal threshold. The MBE is then segmented (*C*) into a series of Minimum Bounding Rectangles (MBRs). The MBRs for candidate *C* are retrieved from the index (*D*). The lower (upper) bound to the true wDTW distance between *Q* and *C* is estimated by the minimum (maximum) distance over both their respective MBRs overlapping in time (*E-F*).

3.2.4.1.1 Lower-bounding the wDTW

There are several DTW lower-bounding techniques in the literature [Yi et al. 1998; Kim et al. 2001; Keogh 2002]. As in Vlachos et al. [2003], Keogh's [2002] method is preferred here since it offers the tightest lower-bound, is readily indexable with MBRs and uses the familiar concept of envelopes [Sakoe and Chiba 1978; Itakura 1975] also used for LCSS indexing. The warping window, specified by the temporal threshold, is used to create a Minimum Bounding Envelope (*MBE*) above and below the query sequence. The MBE for the query Q is defined with regards to its upper envelope U and its lower envelope L such that:

$$MBE_Q = (L, U)$$

$$\text{where} \qquad L_{d,i} = \min_{k=i-\delta_i}^{i+\delta_i} \{q_{d,k}\}$$

$$U_{d,i} = \max_{k=i-\delta_i}^{i+\delta_i} \{q_{d,k}\} \quad \text{for } i \in \{1,\ldots,|Q|\} \text{ and } d \in D$$

δ_i is the temporal threshold at position i in sequence Q with $\delta \in \Re^m$

where $U_{d,i} \geq q_{d,i} \geq L_{d,i}$ which is an obvious but important property of U and L for our lower-bound estimate. The lower-bound is then the squared sum of the distances from every part of the candidate

sequence *C* not falling within the bounding envelope MBE_Q to the nearest edge of the bounding envelope. Hence, the distance between Minimum Bounding Envelope of query *Q* and a candidate *C*, both of dimensionality *D*, is:

$$wDTW(MBE_Q, C) = \sum_{i=1}^{n}\sum_{d=1}^{D} w_d \begin{cases} (c_{d,i} - U_{d,i})^2 & \text{if } c_{d,i} > U_{d,i} \\ (c_{d,i} - L_{d,i})^2 & \text{if } c_{d,i} < L_{d,i} \\ 0 & \text{otherwise} \end{cases}$$

The above formulation of the lower-bound is adapted from that of Keogh [2002] so as to support dimensional weightings introduced with our wDTW function. The lower-bound is illustrated in Figure 47.

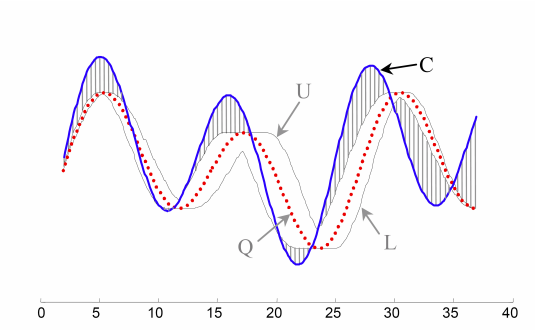

Figure 47: **Lower-bound of the true wDTW distance** between the query *Q* (in *Red*) and candidate *C* (in *Blue*). The Minimum Bounding Envelope, defined by the upper and lower envelopes *U* and *L*, completely encloses the query. The Euclidean distance from every part of *C* not falling within the bounding envelope, to the nearest orthogonal edge of the bounding envelope, is returned as the lower-bound.

During linear scan, this lower-bound can prune off a multitude of expensive wDTW computations with minimal computation costs and with no false negatives [Faloutsos et al. 1994]. It only requires that the MBE is first obtained for the query, which also incurs minimal costs. The method also applies to indexed sequences enclosed in Minimum Bounding Rectangles (MBRs). As we will see, the lower-bound is not as tight but still holds between MBRed sequences.

3.2.4.1.2 Estimating wDTW with MBRs

Our indexing scheme breaks up the query and all database sequences into a series of MBRs. A quick method to estimate the wDTW between MBRed sequences is necessary to quickly prune unpromising candidates. The lower-bounding strategy mentioned above in Section 3.2.4.1.1 has the advantage of also working on two MBRed sequences [Vlachos et al. 2003; Keogh 2002; Roussopoulos et al. 1995; Hjaltason and Samet 1995; Zhu et al. 2003]. A series of MBRs enclosing the query sequence *Q* is obtained from the bounding

envelope MBE_Q (see Section 3.2.5 for details on MBR creation process). The *i*-th *D*-dimensional MBR for Q is defined as the ordered tuple:

$$MBR_{Q,i} = \langle t_{start}, t_{end}, low_1, high_1, \ldots, low_d, high_d, \ldots, low_D, high_D \rangle$$

t_{start} is the starting key-frame of $MBR_{Q,i}$
t_{end} is the final key-frame of $MBR_{Q,i}$
low_d is the lowest detected value over the interval $[t_{start}, t_{end}]$ of MBE_Q for dimension *d*
$high_d$ is the highest detected value over the interval $[t_{start}, t_{end}]$ of MBE_Q for dimension *d*

In the same manner, the *i*-th *D*-dimensional MBR for candidate *C* is defined as the tuple:

$$MBR_{C,i} = \langle t_{start}, t_{end}, low_1, high_1, \ldots, low_d, high_d, \ldots, low_D, high_D \rangle$$

with $low_d = \min\left(c_{d,t_{start}:t_{end}}\right)$ and $high_d = \max\left(c_{d,t_{start}:t_{end}}\right)$. The overall distance estimate between Q and *C* is therefore defined as:

$$E_{\min}^{dtw} = \sum_{i=1}^{n}\sum_{j=1}^{m} MINDIST\left(MBR_{Q,i}, MBR_{C,j}\right)$$

n is the length of the query Q
m is the length of the candidate *C*

The overall distance $E_{\min}^{dtw}$ is the sum of the *MINDIST* function over all MBRs of *C* and Q. $E_{\min}^{dtw}$ underestimates the true distance of both motions with no false-negatives. To support wDTW, we extend the original *MINDIST* function [Roussopoulos et al. 1995; Hjaltason and Samet 1995] to support dimensional weightings:

$$MINDIST\left(MBR_{Q,i}, MBR_{C,j}\right) = \sqrt{\sum_{d=1}^{D} w_d \cdot x_d^2\left(MBR_{Q,i} \bigcap_t MBR_{C,j}\right)}$$

$$where \quad x_d = \begin{cases} \left|MBR_{Q,i}\langle high_d\rangle - MBR_{C,j}\langle low_d\rangle\right| & if \quad MBR_{Q,i}\langle high_d\rangle < MBR_{C,j}\langle low_d\rangle \\ \left|MBR_{Q,i}\langle low_d\rangle - MBR_{C,j}\langle high_d\rangle\right| & if \quad MBR_{C,j}\langle high_d\rangle < MBR_{Q,i}\langle low_d\rangle \\ 0, \ otherwise \end{cases}$$

and where w_d is the weight assigned to dimension *d*, $MBR_{Q,i}\langle high_d \rangle$ denotes element $high_d$ of the tuple $MBR_{Q,i}$, and the operator $\bigcap_t$ measures the total time intersection between two MBRs such that:

$$MBR_{Q,i} \bigcap_t MBR_{C,j} = \min\left(MBR_{Q,i}\langle t_{end}\rangle, MBR_{C,j}\langle t_{end}\rangle\right) - \max\left(MBR_{Q,i}\langle t_{start}\rangle, MBR_{C,j}\langle t_{start}\rangle\right)$$

We supplement the lower-bound estimate $E_{\min}^{dtw}$ with an *upper-bound* estimate $E_{\max}^{dtw}$ on the true distance. The reason for this is that we can prune off sequences with the lower-bound larger than the smallest upper-bound. The overall upper-bound $E_{\max}^{dtw}$ is defined as the sum of the *MAXDIST* function [Roussopoulos et al. 1995; Hjaltason and Samet 1995] over all MBRs of *C* and *Q*. Again, we extend it to support dimensional weightings so that it becomes:

$$E_{\max}^{dtw} = \sum_{i=1}^{n}\sum_{j=1}^{m} MAXDIST\left(MBR_{Q,i}, MBR_{C,j}\right)$$

with

$$MAXDIST\left(MBR_{Q,i}, MBR_{C,j}\right) = \sqrt{\sum_{d=1}^{D} w_d \cdot x_d^2\left(MBR_{Q,i} \bigcap_t MBR_{C,j}\right)}$$

$$\text{where } x_d = \begin{cases} \max\left(MBR_{Q,i}\langle high_d\rangle, MBR_{C,j}\langle high_d\rangle\right) - \min\left(MBR_{Q,i}\langle low_d\rangle, MBR_{C,j}\langle low_d\rangle\right) \\ \quad if \quad \left(MBR_{Q,i}\langle high_d\rangle > MBR_{C,j}\langle high_d\rangle \;\&\; MBR_{Q,i}\langle low_d\rangle > MBR_{C,j}\langle low_d\rangle\right) \\ \quad or \quad \left(MBR_{Q,i}\langle high_d\rangle < MBR_{C,j}\langle high_d\rangle \;\&\; MBR_{Q,i}\langle low_d\rangle < MBR_{C,j}\langle low_d\rangle\right) \\ \max\begin{cases} \max\left(MBR_{Q,i}\langle high_d\rangle, MBR_{C,j}\langle high_d\rangle\right) - \max\left(MBR_{Q,i}\langle low_d\rangle, MBR_{C,j}\langle low_d\rangle\right) \\ \min\left(MBR_{Q,i}\langle high_d\rangle, MBR_{C,j}\langle high_d\rangle\right) - \min\left(MBR_{Q,i}\langle low_d\rangle, MBR_{C,j}\langle low_d\rangle\right) \end{cases} \\ \quad otherwise \end{cases}$$

A more intuitive illustration is given below in Figure 48:

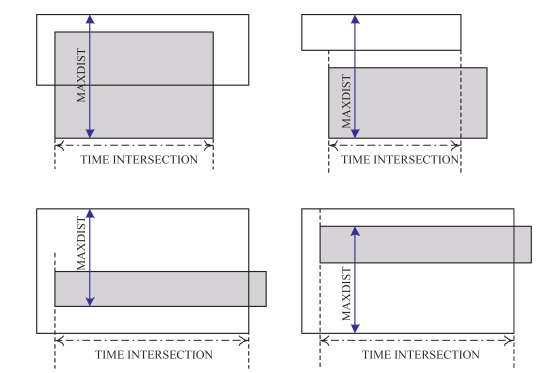

Figure 48: *MAXDIST* function between two MBRs (one in grey, the other in black outline) for one dimension.

3.2.4.1.3 Rapid Subsequence Matching Under wDTW

Having defined our wDTW bounding scheme and estimates, we are now ready to introduce the Nearest Neighbour search algorithm. First, we review the original subsequence matching problem. In subsequence matching, the length of a query sequence is assumed to be generally smaller than all candidate sequences. In other words, given a query sequence Q, and a longer candidate sequence C, the task is to find the subsequence in C which best matches Q under any scaling, and report its offset within C. The resulting brute-force algorithm for wDTW over a database $\mathbf{C}$ is outlined in Table 4. We present it only as a baseline against which we compare our proposed algorithms.

```
function BruteForce_SubsequenceNN(query Q,
                                  temporal threshold δ,
                                  joint weights w)

    BestTimeSeries = null;
    BestMatchVal = Infinity;
    BestOffset = null;
    for i = 1 to |C|, the number of time-series in database C
        for every valid offset in sequence C_i, j = 1 to |C_i|-|Q|
            dist = wDTW_{δ,w}(Q, C_i[j:j+|Q|-1]);
            if dist < BestMatchVal
                BestTimeSeries = i;
                BestMatchVal = dist;
                BestOffset = j;

    return(BestTimeSeries, BestMatchVal, BestOffset)
```

Table 4: Pseudo-code to find the best subsequence match over a database.

The approach in Table 4 is both heavily I/O and CPU demanding. Fortunately, it lends itself well to optimisation using our lower-bounding method. Suppose an index I was generated from database $\mathbf{C}$ during the pre-processing phase. Index I contains a series of MBRs completely enclosing each sequence C. It can be stored either in memory or on disk depending on its size. On the other hand, for the query, its lower-bounding envelope and subsequent enclosing MBRs are generated at *query-time*. Since queries tend to be short (typically 20 to 500 keyframes) computational costs are negligible. The advantage is that the temporal threshold is specified at query time and not during the pre-processing phase, since it is independent of the

index. Using the index, a pre-filtering step aids the quick discovery of close matches to the bounded query. This helps discard distant candidates without using the costly quadratic wDTW algorithm. An improved Nearest Neighbour search algorithm is proposed in Table 5.

```
function LinearLB_SubsequenceNN(query Q,
                                temporal threshold δ,
                                joint weights w,
                                database MBR Index I)
    BestTimeSeries = null;
    BestMatchVal = Infinity;
    BestOffset = null;
    MBE_Q = constructMBE_δ(Q);
    MBR_Q = createMBRs(MBE_Q);

    for i = 1 to |C|, the number of sequences in database C
        MBR_Ci ← I[i]; //retrieve candidate MBRs from memory
        for j = 1 to |MBR_Ci|-|Q|, every valid offset in MBR_Ci
            if MINDIST_w(MBR_Ci[j:j+|Q|], MBR_Q) < BestMatchVal
                C_i ← C[i]; //retrieve original sequence from disk
                dist = wDTW_δ,w(Q,C_i[j:j+|Q|-1]);
            if dist < BestMatchVal //eliminate false alarms
                    BestTimeSeries = i;
                    BestMatchVal = dist;
                    BestOffset = j;

    return(BestTimeSeries, BestMatchVal, BestOffset)function
```

Table 5: Pseudo-code to find the best subsequence match using lower-bound estimates.

The algorithm starts by retrieving the MBRs for each subsequence of some time-series C and computing the lower-bound $MINDIST_w(MBR_{Ci}[j:j+|Q|], MBR_Q)$. If this lower-bound is less than *BestMatchVal*, then we have to call $\mathbf{wDTW}_{\delta,w}$ to compute the true distance and update *BestMatchVal* accordingly. Otherwise, we know that $C_i[j:j+|Q|]$ cannot be less than *BestMatchVal* distant from Q.

LinearLB_SubsequenceNN is an improvement over **BruteForce_SubsequenceNN** as it avoids using wDTW for all the sequences in the database. However, it essentially operates along the same lines since it must retrieve a sequence from disk and execute a full wDTW every time the lower-bound distance is less

than `BestMatchVal`. Instead, we can start by retrieving the time-series in increasing order of their lower-bound value from the index. If the lower-bounds we compute are tight enough, then the best match to the query time-series will be among the first time-series to be retrieved. A more informed use of the bounding envelope is presented in Table 8 (in Appendix C) and is referred to as **`FastLB_SubsequenceNN.`** It follows the standard *MINDIST* and *MAXDIST* based pruning scheme [Roussopoulos et al. 1995; Hjaltason and Samet 1995; Weber et al. 1998]. First, **`FastLB_SubsequenceNN`** estimates the distance between the query envelope and each subsequence in a linear scan fashion. These estimates are then sorted in a queue so that the smallest estimate is at the top. One by one, it pops the queue, checks if the current estimate is below the current upper-bound and lower than `BestMatchVal`, and runs a full distance calculation if necessary. The process returns the best match when the popped estimate is larger than `BestMatchVal` or if the queue is empty. The algorithm performs less work than **`LinearLB_SubsequenceNN`** and only in the worst case does it perform an equal amount of work. The computational cost for the priority queue insertions is $O(\log|\mathbf{C}|)$ which is minimal compared to I/O and CPU costs incurred by full distance calculations. Note that all the approaches presented here can be adjusted to return the k-NN sequences by simply comparing with the *k*-th `BestMatchVal` match [Keogh 2002; Faloutsos and Lin 1995; Roussopoulos et al. 1995].

3.2.4.2 Pruning LCSS calculations

Since the cost of calculating the LCSS is similar to that of wDTW, an efficient pruning scheme is essential. This requires an upper-bound of the true LCSS similarity. This is the opposite situation than with wDTW, where instead of calculating the minimum or maximum distance between the MBRs of the query and the indexed motions, we evaluate the degree of overlap between the MBRs. The steps are outlined in Figure 49.

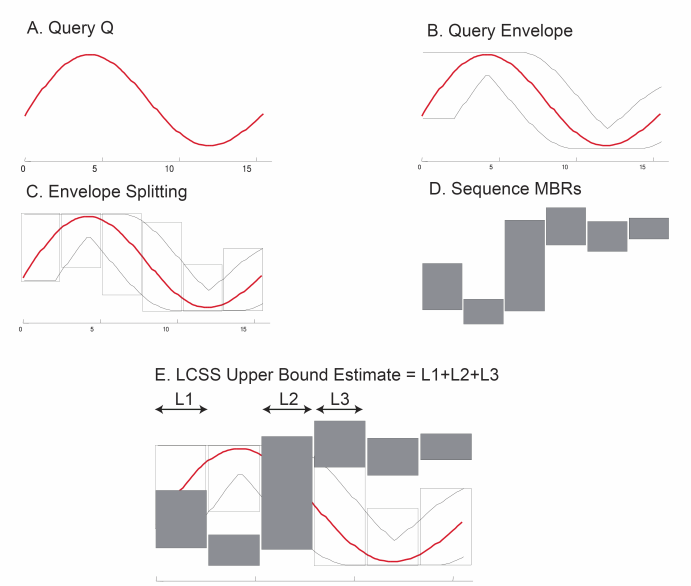

Figure 49: **An illustration of the steps necessary to estimate the LCSS metric** between query *Q* and a candidate *C*. One-dimensional sequences are chosen for clarity. The original user query (*A*) is first enclosed in a Minimum Bounding Envelope (MBE) (*B*) which delineates possible matching areas given LCSS's spatial and temporal thresholds. The MBE is then segmented (*C*) into a series of Minimum Bounding Rectangles (MBRs). The MBRs for candidate *C* are retrieved from the index (*D*). The upper-bound to the true LCSS similarity between *Q* and *C* is estimated by the spatial overlap between both their respective MBRs overlapping in time (*E*).

3.2.4.2.1 Upper-Bounding the LCSS

Vlachos et al.'s [2003] method to upper-bound the true LCSS also uses the concept of Minimum Bounding Envelope (MBE_Q) to enclose the query sequence *Q*. Here, MBE_Q gives the possible matching areas as everything outside this envelope can never be matched. We extend the original formulation of MBE_Q to support time-varying inclusion/exclusion spatial thresholds and time-varying temporal thresholds such that:

$$MBE_Q = (L, U)$$

$$\text{where} \qquad L_{d,i} = \min_{k=i-\delta_i}^{i+\delta_i} \left\{ q_{d,k} + |\varepsilon_{d,i}| \right\}$$

$$U_{d,i} = \min_{k=i-\delta_i}^{i+\delta_i} \left\{ q_{d,k} - |\varepsilon_{d,i}| \right\} \quad \text{for } i \in \{1,\ldots,|Q|\} \text{ and } d \in D$$

The new definition of the LCSS similarity between the query envelope MBE_Q and a candidate *C* is therefore defined as:

$$LCSS(MBE_Q, C) = \sum_{j=1}^{m} \sum_{d=1}^{D} \begin{cases} 1, & \text{if } c_{d,j} \text{ outside envelope when } \varepsilon_{d,j} < 0 \\ & \text{if } c_{d,j} \text{ within envelope when } \varepsilon_{d,j} \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Figure 50 illustrates the process. This value is an upper-bound to the similarity between Q and C such that $LCSS_{\delta,\varepsilon}(MBE_Q, C) \geq LCSS_{\delta,\varepsilon}(Q, C)$. This in turn gives us the *lower-bound* on their distance such as $D_{\delta,\varepsilon}(MBE_Q, C) \leq D_{\delta,\varepsilon}(Q, C)$ with $D_{\delta,\varepsilon}(MBE_Q, C) = 1 - \frac{LCSS_{\delta,\varepsilon}(MBE_Q, C)}{min(|Q|,|C|)}$. The lower-bound guarantees that no points are missed, in other words, that there will be no false negatives.

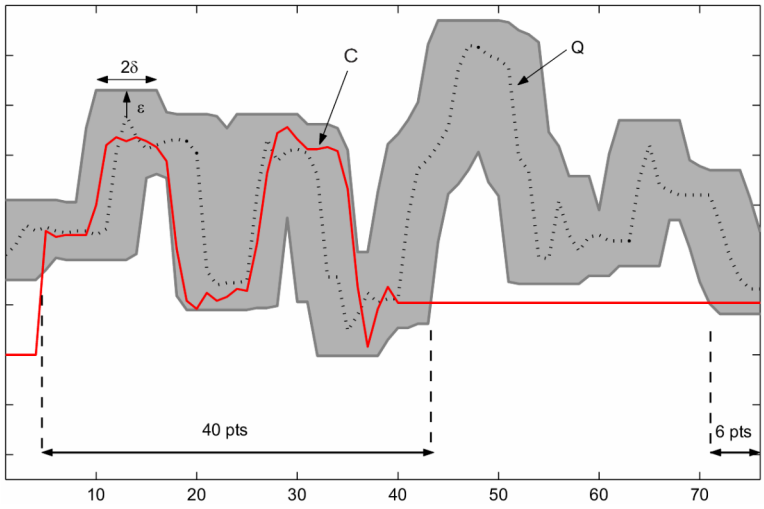

Figure 50: The grey area delineates the **Minimum Bounding Envelope** (MBE_Q) enclosing the query sequence Q. The LCSS similarity estimate with the candidate sequence C, in red, totals to 46. It is the number of points from C lying inside the envelope defined by its constant spatial and temporal thresholds ε and δ.

3.2.4.2.2 Estimating LCSS with MBRs

The similarity estimation for the LCSS [Vlachos et al. 2002] process is generally the same as for wDTW. The upper-bound is obtained by evaluating the amount of spatial overlap over each dimension for any temporally intersecting MBRs from both sequences. The overall upper-bound E^{lcss} is defined as the sum of all the temporally and spatially intersecting MBRs of C and Q such that:

$$E^{lcss} = \sum_{i=1}^{n} \sum_{j=1}^{m} \left(MBR_{Q,i} \bigcap_t^o MBR_{C,j} \right)$$

where

$$MBR_{Q,i} \bigcap_t^o MBR_{C,j} = \sum_{d=1}^{D} \begin{cases} MBR_{Q,i} \bigcap_t MBR_{C,j} \\ \quad if \quad \left(MBR_{Q,i}\langle low_d \rangle < MBR_{C,j}\langle high_d \rangle \;\&\; MBR_{Q,i}\langle high_d \rangle > MBR_{C,j}\langle low_d \rangle \right) \\ 0 \\ \quad otherwise \end{cases}$$

The first condition in $MBR_{Q,i} \bigcap_t^o MBR_{C,j}$ ensures that only MBRs that intersect spatially on the given dimensions, get to increment the global estimate by their temporal overlap.

3.2.4.2.3 Rapid Subsequence Matching Under LCSS

The subsequence algorithm under LCSS operates in a similar manner as its equivalent under wDTW. The difference is that we use the upper-bound to prune matches. We therefore only show the final optimised **`FastUB_SubsequenceNN`** algorithm in Table 10 (in Appendix C) which uses a priority queue and the upper-bound estimate E^{lcss}.

3.2.5 Index Generation

Our indexing schemes approximate each motion by a sequence of minimum bounding (hyper-) rectangles (MBR). We can represent a motion using its bounding box in space and time. However, this creates large amounts of empty space and consequently a poor distance estimate. Instead, a better approximation of the original motion is obtained by segmenting it into multiple consecutive and tighter MBRs minimizing the total volume consumption (Figure 51). Each split along the time axis is chosen so as to minimize the total volume consumption. The reason for this is that minimizing the total volume occupied by MBRs of two sequences minimizes their expected similarity estimate. While using more temporal splits to approximate each motion improves query performance by reducing the empty space, every split corresponds to a new MBR data structure and thus increases the index memory requirements and the cost of estimate calculations.

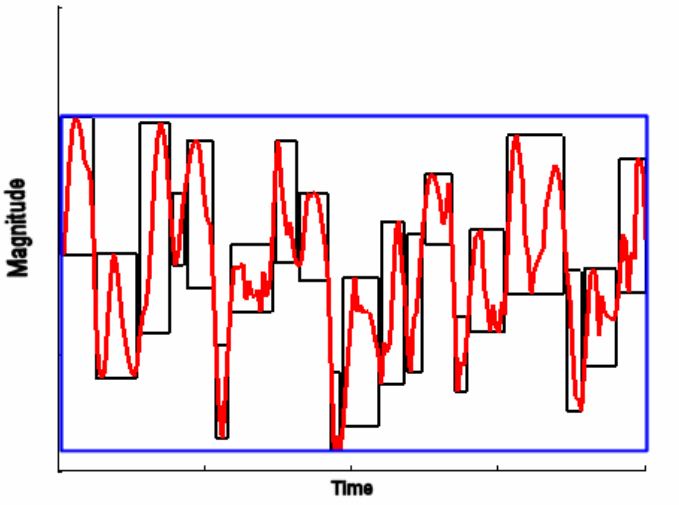

Figure 51: **MBR Segmentation**. One-dimensional sequence (*in red*) enclosed by one MBR (*in blue*) introduces a lot of empty space. The approximation is much tighter when enclosed in 20 smaller MBRs (*in black*).

We therefore use a value for the number and location of splits that achieve a good trade-off between estimate accuracy, and space/computational overhead. Each sequence in the database is assigned a total of $a \cdot |C|$ fixed-length splits. The $O(|C|\log(|C|))$ algorithm starts with one MBR per key-frame and merges all consecutives MBRs that give the smallest volume increase [Hadjieleftheriou et al. 2002]. Figure 52 shows the segmented query Minimum Bounding Envelope (MBE) and a candidate sequence. Note that the no false negative property is guaranteed regardless of the number of splits.

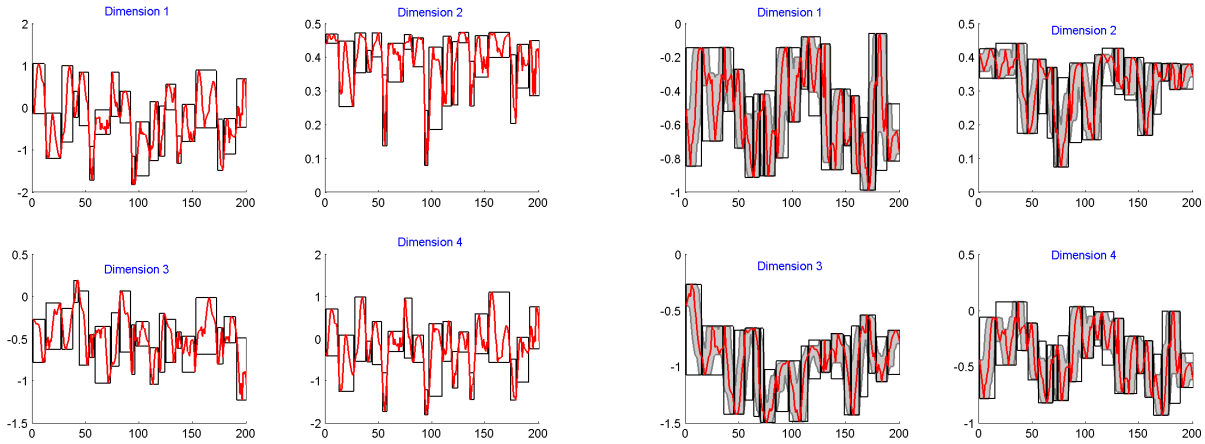

Figure 52: **MBR Generation over Data**. (*Left*) Segmentation of a 4-dimensional sequence composed of 200 frames with $a = 0.1$. The sequence is completely contained in 20 MBRs. (*Right*) Segmentation of a query's minimum bounding envelope (MBE) for a 4-dimensional query sequence.

3.2.6 Uniform Scaling Support

Humans typically match two mocap sequences of different length by globally stretching or shrinking them to the same length. They are then compared locally key-frame-by-key-frame, with some warping within a small neighbourhood in the time axis. The two-step transform therefore begins with *uniform time warping* (i.e. global/uniform scaling) followed by *local time warping* [Park et al. 2000]. In other words, the speed of both mocap sequences are first matched, after which small time variabilities inherent to mocap are removed.

As noted in Section 3.2.3.2 , the Euclidean distance has difficulty dealing with sequences of different lengths and is very sensitive to small variations in the time axis. On the other hand, the DTW and LCSS distances address the problem of *local time warping* by only considering local adjustments of the time axis rather than

global adjustments. In this section, we introduce scale-independent matching by supporting uniform time warping over the Euclidean, DTW and LCSS distances. To illustrate the difference between DTW and uniform scaling, we record a one second snippet of an individual's electrocardiogram twice. On the first occasion, the time-series captures two heartbeats, while on the second (during exercise), it captures three heartbeats. When we use DTW to measure the similarity of the two sequences, we get meaningless results because DTW must match every point and there is simply no sensible way of mapping two heartbeats to three (Figure 53 (left)). By contrast, uniform scaling can stretch the faster heartbeat until a near perfect alignment is found (Figure 53 (right)). As mentioned above, DTW can be useful to remove subtle local differences *after* uniform scaling has located the best global match [Faloutsos et al. 1994].

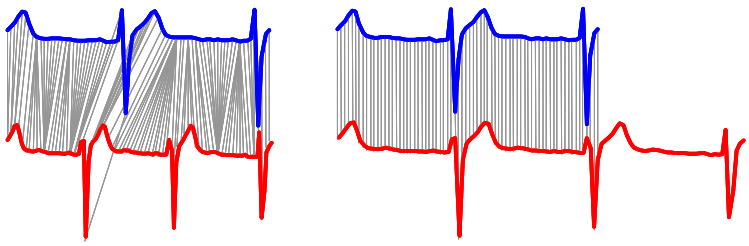

Figure 53: A visual contrast of DTW (*Left*) and Uniform Scaling (*Right*). Using DTW we cannot achieve an intuitive alignment between two and three heartbeats, even if the sequences happen to be of the same length. However uniform scaling, by stretching the bottom sequence, achieves a meaningful alignment.

The importance of scale-independent matching has also been observed in other domains such as bio-informatics [Aach and Church 2001], speech recognition [Rabiner and Juang 1993], gait analysis [Gavrila and Davis 1995] and music retrieval [Zhu and Shasha 2003]. Traditionally, supporting uniform scaling is computationally expensive. Previous work [Chu et al 1998; Park et al. 2000; Kahveci and Singh 2001] has attempted to speed up matching under uniform scaling. However, all previous work has focused on speeding up the similarity search, when the scaling factor is known. The feature that differentiates our method from other work is that a user is allowed to issue a single query and find the best match at any scaling.

There exists one other technique in the literature that allows similarity search under uniform time scaling while guaranteeing no false negatives, namely the *CD-criterion* technique of Argyros and Ermopoulos [2003]. The algorithm works by solving an inequality determining whether the Euclidean distance between two sequences under any scaling is under a given threshold or not. While pioneering, we do not see this work as a complete solution to our problem for the following reasons. The algorithm can only test if sequences are within a user-supplied epsilon, and thus cannot be used for ranking, classification or clustering. The

algorithm requires a parameter to be set; this parameter does not affect the accuracy, but it can affect the speedup. The real weakness of the approach is that it only speeds up main memory search, and cannot be indexed. In fact, the authors suggest that indexing of uniform scaling "appears unfeasible" [Argyros and Ermopoulos 2003], although this is exactly the contribution of the current work.

We formalize the uniform scaling problem in Section 3.2.6.1 and describe a way to considerably speed up scale-invariant matching using a lower-bounding technique in Section 3.2.6.2 . The concept is then extended to subsequence matching in Section 3.2.6.4 .

3.2.6.1 Uniform Scaling Definition

For notational simplicity, we assume that the query Q is always shorter or equal to the candidate C, i.e. $n \le m$. Consequently, we only consider stretching the query to match some prefix of C. Support for shrinking is possible by first down-sampling the query by the targeted shrinking factor and then appropriately re-adjusting the original stretch factor. For example, if we want to perform a query of length 100 with the flexibility to shrink or stretch by 10%, the system interpolates the query down to 90 keyframes and then searches for matches stretched up to 122.22%. In other words, given that $stretch = 0.1$ and $shrink = 0.1$, the down-sampled query has a new length $n' = n \cdot shrink = 100 \cdot (1 - 0.1) = 90$ and the new scaling factor is $stretch' = \frac{1 + stretch}{1 - shrink} = \frac{1 + 0.1}{1 - 0.1} \approx 1.22$.

The Euclidean distance between C and Q is only defined when $m = n$. In our case, $n \le m$. Therefore some elements are left unmatched. To compare the two time-series in this case, we have several choices; we can truncate C, and compare Q to $((c_{1,1},\ldots,c_{d,1},\ldots,c_{D,1}),\ldots,(c_{1,n},\ldots,c_{d,n},\ldots,c_{D,n}))$, or we can somehow stretch Q to be of length m, or more generally we can stretch Q to be of length p, ($n \le p \le m$), truncate off the last m-p values of Q, then use the Euclidean distance [Keogh 2003]. The informal idea behind stretching can be captured in the more formal definition of scaling. To scale time-series Q to produce a new time-series QP of length p, the formula is:

$$QP_j = Q_{\lceil j \cdot n/p \rceil}, 1 \le j \le p$$

where p/n is referred to as the scaling factor (*sf*). To find the best scaled match between the query and the candidate, we have to test all possible scalings as shown in Table 6.

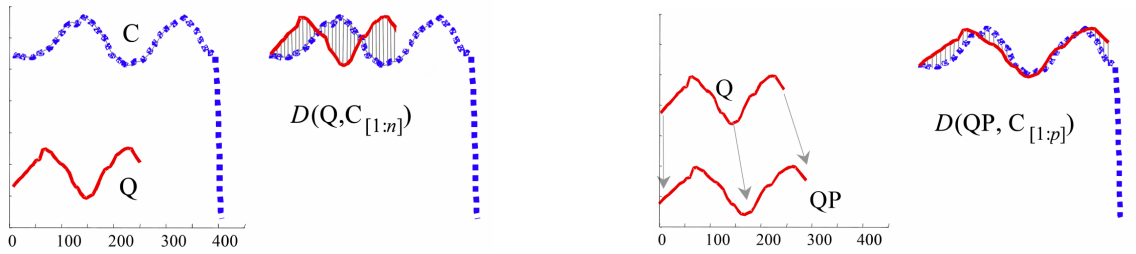

Figure 54: **Uniform Scaling under Euclidean Distance**. (*Left*) The candidate *C* in blue is longer than the query *Q* in red. Their Euclidean distance is illustrated next to the figure. (*Right*) The query *Q* is stretched to *QP* of length *p* producing a good match to the first *p* keyframes in *C*.

For whole sequence matching, the **`TestAllScalings`** function in Table 6 is executed for each candidate sequence in the database $\mathbf{C}$. We can interchangeably select $d_{wEuclid}$ or wDTW as the distance function $d(.)$. The LCSS distance can also be introduced with minor changes similar to those in Section 3.2.4.2.3. Since wDTW and LCSS inherently support matching between sequences of different lengths, there is no need to up-sample the query. Hence in Table 6, the line `QP = rescale(Q,p);` is replaced by `QP = Q;`. The resulting brute force algorithm is outlined in Table 7. It has an inflated time complexity of $O\left(|\mathbf{C}|\cdot(m-n)\right)$ under even the fastest distance function, *i.e.* the weighted Euclidean distance.

```
function TestAllScalings(Q,C)

      BestMatchVal = Infinity;
      BestScalingFactor = null;
      for p = n to m
            QP = rescale(Q,p);
            dist = d(QP, C1:D,1:p);
            if dist < BestMatchVal
                  BestMatchVal = dist;
                  BestScalingFactor = p/n;

      return(BestMatchVal, BestScalingFactor)
```

Table 6: Pseudo-code to find the best-scaled match between two sequences

```
function SearchDatabaseforScaledMatch(Q)

      OverallBestTimeSeries = null;
      OverallBestMatchVal = Infinity;
      OverallBestScaling = null;
            for i = 1 to |C|, the number of sequences in database C
            [dist, scale] = TestAllScalings(Q,C_i)
            if dist < OverallBestMatchVal
                  OverallBestTimeSeries = i;
                  OverallBestMatchVal = dist;
                  OverallBestScaling = scale;

      return(OverallBestTimeSeries,OverallBestMatchVal,OverallBestScaling)
```

Table 7: Pseudo-code to the find best-scaled whole match over a database.

3.2.6.2 Lower-Bound to Uniform Scaling

In order to cut down on the number of executions of the **TestAllScalings** function, a lower-bound estimate of the true distance under any possible scaling is introduced. It works in much the same way as the lower-bound for DTW or LCSS, where a fast lower-bounding function is used to help prune sequences that could not possibly be the best match. The current scaling factor is used to create a now familiar Minimum Bounding Envelope (*MBE*) above and below each *candidate* sequence. With wDTW and LCSS, the envelope is created from the query, not the candidate. Therefore, the MBE for a candidate *C* is defined as:

$$USMBE_C = (L, U)$$

where

$$L_{d,i} = \min_{k=\lfloor (i-1)\cdot m/n \rfloor + 1}^{\lfloor i\cdot m/n \rfloor} \{c_{d,k}\}$$

$$U_{d,i} = \max_{k=\lfloor (i-1)\cdot m/n \rfloor + 1}^{\lfloor i\cdot m/n \rfloor} \{c_{d,k}\} \quad for\ i \in \{1,\ldots,|Q|\}\ and\ d \in D$$

$U_{d,i}$ is the upper envelope value at key-frame *i* for dimension *d*

$L_{d,i}$ is the lower envelope value at key-frame *i* for dimension *d*

sf_{max} is the maximum scaling factor, where $sf_{max} = 1$ means no scaling

$n = |Q|$

$m = n \cdot sf_{\max}$

The $USMBE_c$ bounds the first m keyframes of candidate C under all scalings of the query between $[1, sf_{max}]$. Figure 55 illustrates the bound. It can be shown that the distance between the candidate's envelope and the query lower-bounds the true distance between Q and C under uniform scaling [Keogh 2003]. For scale-independent matching with the Euclidean distance, this signifies that:

$$d_{wEuclid}(Q, USMBE_C) \le d_{wEuclid}(QP, C[1:p])$$

QP is the rescale query Q for any scale factor $sf \in [1; sf_{max}]$

$p = |QP| = sf \times |Q|$

$C[1:p]$ are the first p keyframes of candidate C

Similarly, wDTW (LCSS) is also lower (upper) bounded under uniform scaling such that:

$$wDTW(MBE_Q, USMBE_C) \le wDTW(Q, USMBE_C) \le wDTW(Q, C[1:p]) \text{ and}$$
$$LCSS_{\delta,\varepsilon}(MBE_Q, USMBE_C) \ge LCSS_{\delta,\varepsilon}(Q, USMBE_C) \ge LCSS_{\delta,\varepsilon}(Q, C[1:p])$$

where MBE_Q is the minimum bounding envelope for the query Q either under wDTW or LCSS. There is no need to rescale the query Q since wDTW and LCSS support sequences of different lengths.

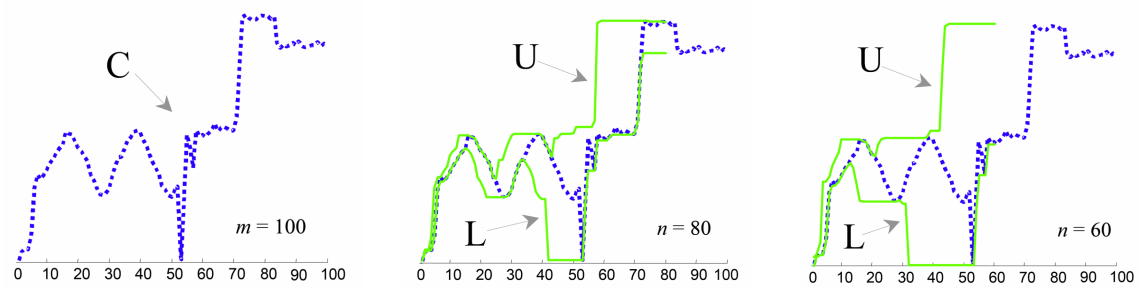

Figure 55: **Extracting the scaling envelope**. (*Left*) One-dimensional candidate sequence C of length $|C| = 100$. (*Middle*) The upper and lower envelopes U and L of length 80 completely enclose all uniformly scaled versions of C. (*Right*) Candidate C is now enclosed by U and L of length 60.

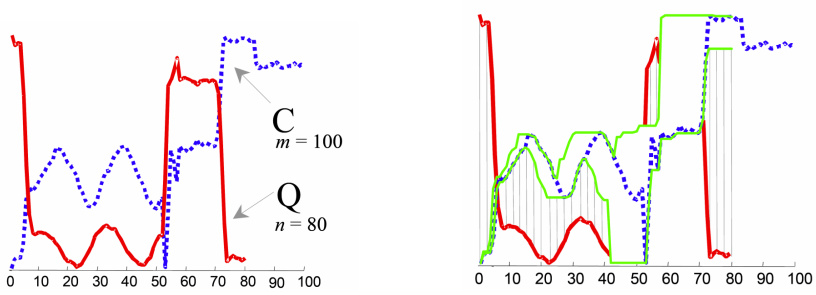

Figure 56: **Bounding the Euclidean distance under uniform scaling**. (*Left*) The query Q is shorter than candidate C. (*Right*) The lower-bounding distance $d_{wEuclid}(Q,USMBE_C)$ corresponds to the sum of the squared straight line distances from the query to the nearest point in the envelope (in grey). Any part of the query that falls inside the envelope is ignored.

3.2.6.3 Estimating Uniform Scaling with MBRs

The above bounding technique requires that we create an envelope for each candidate sequence in the database. We do this beforehand during the database pre-processing phase. A series of MBRs are extracted from each candidate envelope using techniques presented in Section 3.2.5. The distance or similarity estimation process is essentially the same with or without uniform scaling. First, the query envelope and subsequent MBRs are computed at query time. The overlap (or distance) between MBRs of the query and each candidate dictates candidate pruning. The full distance is then calculated over the retained candidates using the **`TestAllScalings`** function. This gives us the best match along the corresponding scaling factor.

Since the uniform scaling enveloping takes place during the pre-processing phase, we are locked into a preset scaling factor. In practice, this is not a problem since we will typically want to limit the amount of scaling anyway. We preset upper and lower limits on the scaling factor based on limitations of human biomechanics. For example, most people can only speed up their natural walk about 20% before changing their gait into a run [Gavrila and Davis 1995]. If the database was created to allow 20% scaling, and the user now wanted to do 25% scaling, then the database would have to be reprocessed. However, if the database was created to allow up to 20% scaling, and the user only wanted up to 10% scaling, we can use the same index and prune off the false hits. Although we did not have to rebuild the index, the performance would not be as good as if the index was purpose-built for exactly 10%. The reason for this is that the scaling bound is not as tight as it could be for a smaller scaling range. One solution is to have multiple indexes with each one doubling the range, *i.e.* an index for up to 5% scaling, one for 10% and one for 20%.

3.2.6.4 Uniform Scaling over Subsequence Matching

The lower-bounding approach outlined above, developed in collaboration with Eamonn Keogh and Michalis Vlachos of the University of California, is only defined for whole sequence matching. This section extends it to handle subsequence matching. In subsequence matching, the length of a query sequence is assumed to be generally smaller than all candidate sequences. In other words, given a query sequence *Q*, and a longer candidate sequence *C*, the task is to find the subsequence in *C*, beginning at C_j which best matches *Q* under any scaling and report its offset within *C*. The resulting brute force algorithm **`BruteForce_ScaledSubsequenceNN`** for wDTW over a database **C** is outlined in Table 10 (in Appendix C). It is untenable over large databases.

We can dramatically speed up the similarity search by extending our existing uniform scaling bounding scheme to subsequence matching. The idea is to calculate a uniform scaling envelope for every subsequence offset along the candidate sequence, and re-enclose them into a single *global* bounding envelope. Then, any portion of this single envelope bounds the subsequence under any uniform scaling within a pre-defined range. This range is defined at index creation time and corresponds to sf_{max}. In other words, the *local* uniform scaling envelope $USMBE_{C[j:j+|Q|\cdot sf_{max}-1]}$ (see below) is obtained for every subsequence offset *j* along *C* and merged into a single *global* bounding envelope $USMBE_C^{sub}$ for candidate *C* (Figure 57 and Figure 58). The global scaling envelope $USMBE_C^{sub}$ is defined as:

$$USMBE_C^{sub} = (L, U)$$

$$\text{where} \qquad \text{for } i \in \{1,\ldots,|Q|\} \text{ and } d \in D,$$

$$L_{d,i} = \min_{a=\lfloor (i-1)\cdot m/n \rfloor+1,\, b=\lfloor i\cdot m/n \rfloor}^{a=\lfloor (i+|C|-n-2)\cdot m/n \rfloor+1,\, b=\lfloor (i+|C|-n-1)\cdot m/n \rfloor} \min_{k=a}^{b} \{c_{d,k}\}$$

$$U_{d,i} = \max_{a=\lfloor (i-1)\cdot m/n \rfloor+1,\, b=\lfloor i\cdot m/n \rfloor}^{a=\lfloor (i+|C|-n-2)\cdot m/n \rfloor+1,\, b=\lfloor (i+|C|-n-1)\cdot m/n \rfloor} \max_{k=a}^{b} \{c_{d,k}\}$$

with $n = |Q|$ and $m = n \times sf_{\max}$. The $USMBE_C^{sub}$ bounds all subsequences of candidate *C* under any scaling in $[1, sf_{max}]$. The distance between any subsequence envelope and the query lower-bounds the true distance between *Q* and *C* under any valid scaling. For the Euclidean distance, this signifies that:

$$d_{wEuclid}\left(Q, USMBE_C^{sub}\left[j:j+|Q|-1\right]\right) \le d_{wEuclid}\left(Q, USMBE_{C\left[j:j+|Q|\cdot sf_{max}-1\right]}\right) \le d_{wEuclid}\left(QP, C\left[j:j+p-1\right]\right)$$

j is an offset in candidate C

$USMBE_C^{sub}\left[j:j+|Q|-1\right]$ is the *global* scale-bounded subsequence of candidate C at offset j

$USMBE_{C\left[j:j+|Q|\cdot sf_{max}-1\right]}$ is the *local* scale-bounded subsequence of candidate C at offset j

sf_{max} is the maximum scaling factor, where $sf_{max}=1$ means no scaling

QP is the rescale query Q for any scale factor $sf \in [1; sf_{max}]$

$p=|QP|=sf\cdot|Q|$ is the end position of subsequence match in candidate C given a scale factor sf

$C[1:p]$ are the first p keyframes of candidate C

The bound $USMBE_C^{sub}\left[j:j+|Q|-1\right]$ is not as tight as $USMBE_{C\left[j:j+|Q|\cdot sf_{max}-1\right]}$, since the former encloses bounds from scaled subsequences at earlier offsets. This will inevitably lead to more false alarms but not affect the no-false-negative property of our index. In a similar manner, wDTW is also lower-bounded under uniform scaling such that:

$$wDTW\left(MBE_Q, USMBE_C^{sub}\left[j:j+|Q|-1\right]\right) \le wDTW\left(MBE_Q, USMBE_{C\left[j:j+|Q|\cdot sf_{max}-1\right]}\right) \quad \text{and}$$

$$wDTW\left(MBE_Q, USMBE_{C\left[j:j+|Q|\cdot sf_{max}-1\right]}\right) \le wDTW\left(Q, USMBE_{C\left[j:j+|Q|\cdot sf_{max}-1\right]}\right) \le wDTW\left(Q, C\left[j:j+p-1\right]\right)$$

For the case of LCSS, we get:

$$LCSS_{\delta,\varepsilon}\left(MBE_Q, USMBE_C^{sub}\left[j:j+|Q|-1\right]\right) \ge LCSS_{\delta,\varepsilon}\left(MBE_Q, USMBE_{C\left[j:j+|Q|\cdot sf_{max}-1\right]}\right) \quad \text{and}$$

$$LCSS_{\delta,\varepsilon}\left(MBE_Q, USMBE_{C\left[j:j+|Q|\cdot sf_{max}-1\right]}\right) \ge LCSS_{\delta,\varepsilon}\left(Q, USMBE_{C\left[j:j+|Q|\cdot sf_{max}-1\right]}\right) \ge LCSS_{\delta,\varepsilon}\left(Q, C\left[j:j+p-1\right]\right)$$

The slow **`BruteForce_ScaledSubsequenceNN`** algorithm is dramatically speeded up using our scaling bound as shown in Chapter 5. The ensuing scale-bounded matching algorithms are essentially the same as the ones presented in Section 3.2.3 for weighted Euclidean, wDTW and LCSS metrics. The first difference is that the true distance function is replaced with the appropriate variant of **`TestAllScalings()`**. The second difference is in the index creation phase, where the scaling envelope $USMBE_C^{sub}$ is first evaluated over each candidate before the final MBRs are extracted. A faster way to obtain the global envelope is to evaluate only a chosen subset of the local envelopes along offsets a_n rather than at every offset, such that:

$$a_{n+1} = a_n + \frac{1}{2}\left(|C|-a_n\right)\left(1-sf_{max}\right) \qquad \text{with} \quad a_0 = 1; \forall n, a_n \le |C|$$

This is because only the envelopes that are defined beyond the end of the last envelope contribute to the final global envelope. Figure 59 illustrates the process.

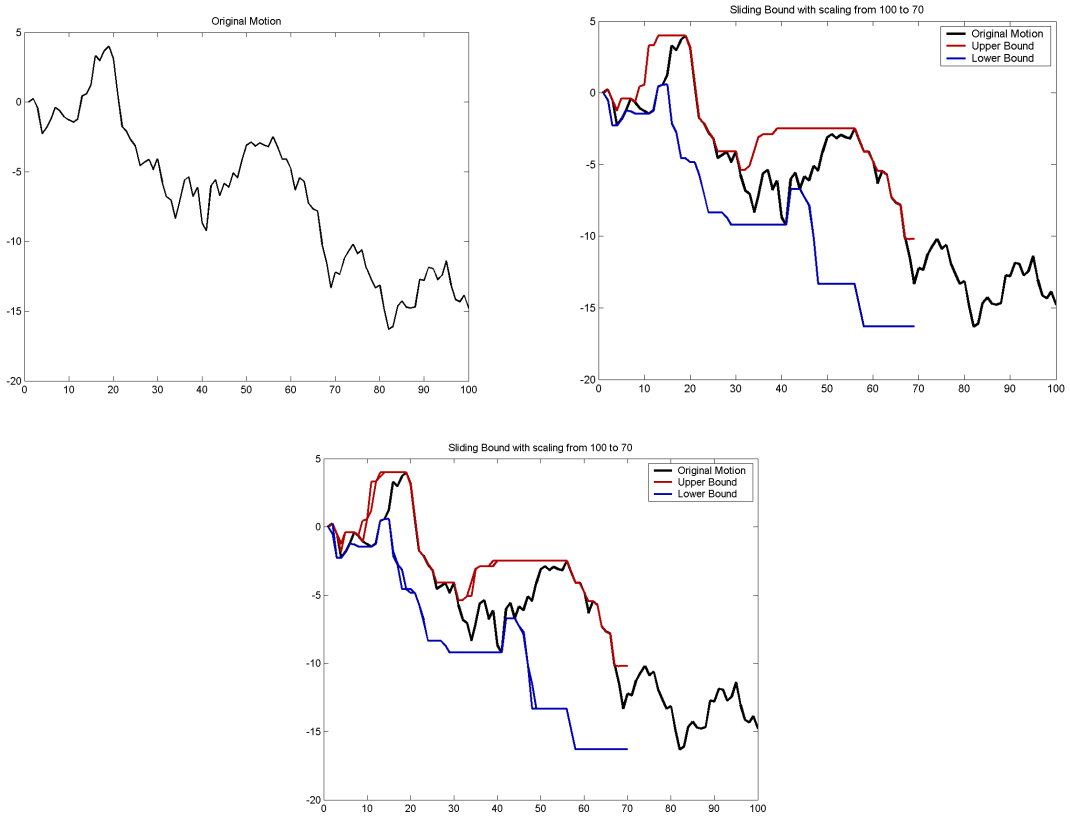

Figure 57: **Subsequence Scaling Bound Creation Steps *Part 1.*** (*Top-Left*) We start off with a one-dimensional candidate sequence C of length 100, although the enveloping process occurs simultaneously on all dimensions of the candidate. (*Top-Right*) The candidate *C* is shrouded by the upper (in red) and lower (in blue) envelopes with lengths 70. This bounds all query stretches of up to 142% starting at position *C*[1]. In other words, we have just calculated $USMBE_{C[j:j+|Q|\cdot sf_{max}-1]}$ such that $USMBE_{C[1:1+|Q|\cdot 1.42-1]}$ (*Bottom*) Now, we slide right by one key-frame in the timeline and recalculate the scaling envelope. The latter bounds all query stretches starting from offset *C*[2], i.e. $USMBE_{C[2:2+|Q|\cdot 1.42-1]}$. As depicted in Figure 58, we proceed in this fashion until the end of the candidate sequence is reached.

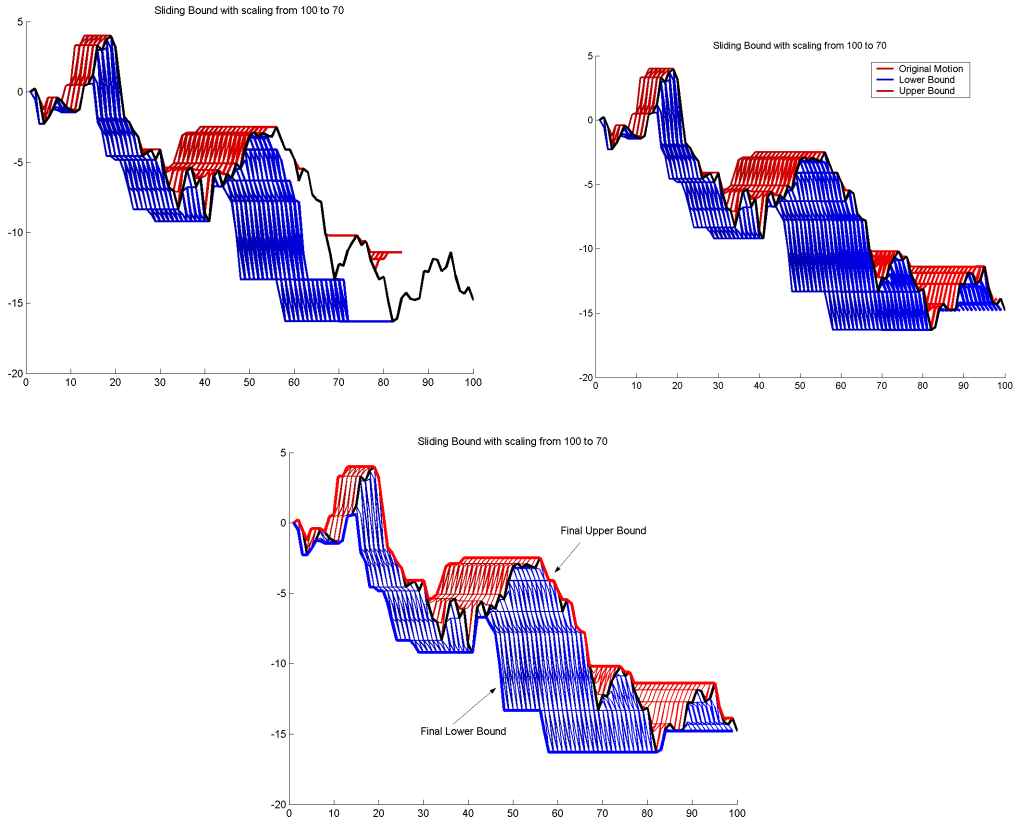

Figure 58: **Subsequence Scaling Bound Creation Steps *Part 2.*** (*Top-Left*) We follow on from Figure 57 by calculating the envelope for every offset in candidate *C*, until we reach the end (*Top-Right*). (*Bottom*) Finally, the global upper and lower-bounds for any subsequence for query stretches of 142% are obtained by enclosing all *sub-envelopes*. They are stored in $USMBE_C^{sub}$ and inserted in the index.

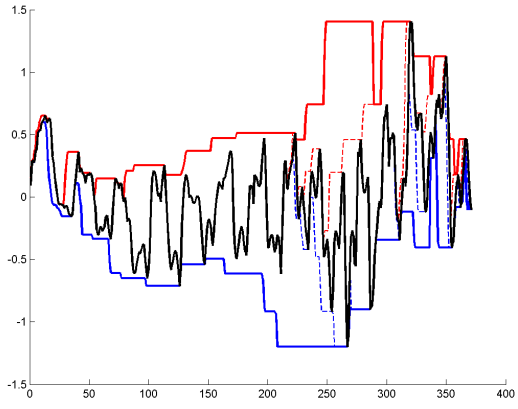

Figure 59: **Quick subsequence scale bounding scheme**. We only calculate the scaling bound at chosen offsets (in dotted) along candidate *C* (in black). The final global envelope (in bold red and blue) shrouds all generated local envelopes.

3.2.7 Optimized Full-Body Matching

To conduct matching over the whole body with the angle-axis joint representation, one has to include all 17 joints (51 dimensions) in the matching area. Performance degrades as the number of dimensions increases. This is due to the exponential growth of the MBR as a function of dimensionality [Aggarwal 2001], consequently affecting the tightness of the distance estimates. One well-known method for improving performance is to use a dimensionality reduction method. The data is transformed to a lower-dimensional space by finding a new axis system in which most of the data variance is preserved in a few dimensions. Many techniques exist for time-series mining, including the Discrete Fourier Transform (DFT) [Agrawal et al. 1993], several kinds of Wavelet transforms (DWT) [Chan and Fu 1999], Piecewise Aggregate Approximation (PAA) [Yi and Faloutsos 2000] and Principal Component Analysis (PCA) [Korn et al. 1997]. We chose PCA since it is the optimal dimensionality reduction method under the Euclidean metric and has been used successfully over motion data [Bowden 2000; Brand and Hertzman 2000; Molina-Tanco and Hilton 2001; Sidenbladh et al. 2002; Bevilacqua 2002; Tanaka and Uehara 2003].

In the following section, we describe how we compress the motion using PCA and why it results in faster matching.

3.2.7.1 Principal Component Analysis

We will now briefly describe the PCA method [Jollife 1986]. Suppose we represent our whole mocap database as N vectors $\mathbf{x}$, where N is the total number of keyframes.. Each vector $\mathbf{x}$ describes a D-dimensional body configuration for a single key-frame. We could easily describe each vector $\mathbf{x}(t)$ as a sum of D basis vectors of dimension D, for example the unit vectors in each direction. If we let $\mathbf{z}_i$ represent basis vector i, and $a_i(t)$ represent the corresponding coefficient at key-frame t, we could then write $\mathbf{x}(t) = \sum_{i=1}^{D} a_i(t)\mathbf{z}_i$. However, that may encompass more information than is needed. There often exists a strong correlation between measurements along different axes in the data set, as we expect for the joint angle or positional mocap data. Therefore, a good representation of each vector $\mathbf{x}(t)$ can be achieved by using a set of M basis vectors of length D, where $M < D$. The goal of PCA is to find the set of such vectors that provides the best possible representation of the data (Figure 60). In other words, we want to be able to write $\mathbf{x}(t)$ as:

$$\mathbf{x}(t) = \sum_{i=1}^{M} a_i(t)\mathbf{z}_i + \sum_{i=M+1}^{D} b_i \mathbf{z}_i$$

where the b_i are constants. The optimal approximation under the Euclidean metric is achieved by choosing the *M* basis vectors to be the eigenvectors of the covariance matrix *C* of the data set having the largest eigenvalues. Let $\bar{\mathbf{x}}$ be the mean of the data set and

$$C = \sum_{t=1}^{N} [\mathbf{x}(\mathbf{t}) - \bar{\mathbf{x}}][\mathbf{x}(\mathbf{t}) - \bar{\mathbf{x}}]^T$$

We can then find the D eigenvectors $\mathbf{z}_i$ of *C* such that:

$$C\mathbf{z}_i = \lambda_i \mathbf{z}_i$$

where λ_i are the eigenvalues. PCA therefore projects the mocap data into a linear subspace with a minimum loss of information. Note that if *M=D*, the original data is exactly reconstructed. The coefficients a_i for representing $\mathbf{x}$ as a sum of these *M* vectors may be found by using:

$$a_i(t) = \mathbf{z}_i{}^T \mathbf{x}(t)$$

Finally, the constants b_i can be found using:

$$b_i = \mathbf{z}_i{}^T \bar{\mathbf{x}}$$

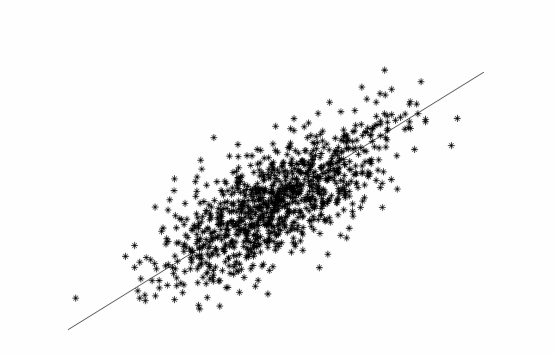

Figure 60: **Principal Component analysis of a two dimensional data cloud**. The line shows the direction of the first principal component that gives an optimal linear reduction of dimension from 2 to 1.

3.2.7.2 A Compact Body Representation

The potential dimensionality contained in motion capture data far exceeds the true dimensionality of human motion. There is considerable redundancy in the representation since not all combinations of physically plausible DoFs are used. With angle-axis body parameterisation, all joints feature three rotational DoFs when fewer are typically required: one for the elbow and knee and two for the hands. In addition, joints have biomechanical limits in their range. With the positional parameterisation, each joint's coordinates compound all the positional information from its parent.

By analysing the magnitude of the corresponding eigenvalues, the minimum dimensionality of the space on which the data lies can be calculated and the information loss estimated. Figure 61 shows the spectrum of the principal components (PCs) over the angle-axis and positional body parameterisation. It shows the contribution of each of the PCs over the whole database.

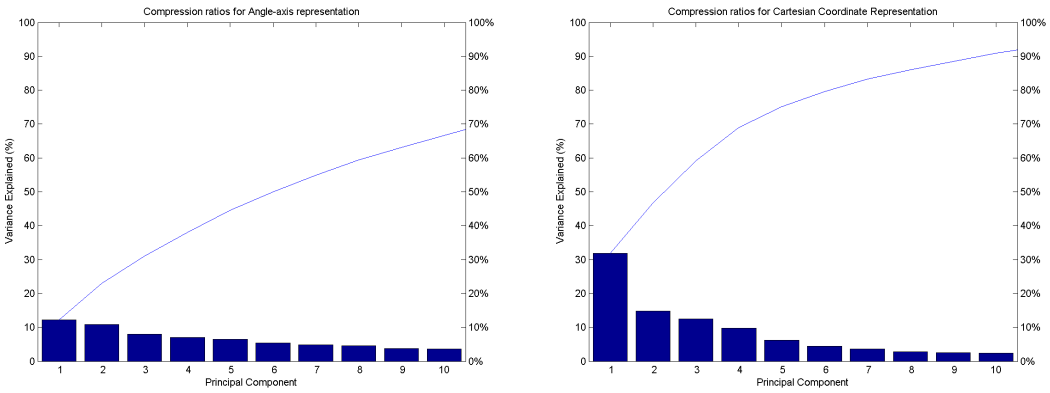

Figure 61: **Principal Component Spectrum**. The x-axis shows the eigenvectors, i.e. principal components, ordered by descending contributions. The y-axis shows how much variance each PC contributes over all the dataset. (*Left*) Plot for the angle-axis body parameterisation. (*Right*) Plot for the positional body parameterisation.

Notice how the spectrum drops off quickly for the positional parameterisation, allowing a good summary of the data with only a few dimensions. The effect is not as pronounced with the angle-axis parameterisation, even though it requires fewer values to parameterize the body than its positional counterpart. This is to be expected since each joint is given the same weighting in the PCs. The hand rotation will have as much impact on the biggest eigen-valued PC as the hip rotation. Minor joints are thus influencing the principal component analysis whilst differences in PCs are not corresponding to differences in visual similarity. Including pre-defined matching joint weights is futile since they would be factored out during the PCA pre-processing step. The problem does not occur with the highly redundant positional parameterisation because the relative importance of each joint is hierarchically factored into the coordinate data at each joint.

Therefore, a difference in hand position has little impact on the main eigenvector; the bulk of the mode is taken up by more important joints such as the hip and shoulders. Unsurprisingly, only the first 15 eigenvectors are required to represent 97.2% of the variation in the database, compared to 32 eigenvectors for the angle-axis representation. Note that we always ignore the global translation component in the PCA representation due to its overwhelming and unwanted influence on the PCA. We can add global translation as additional DoFs later during the matching process if required. Figure 62 shows the first 8 eigenvectors for the positional representation. The first few modes (i.e. PC) of the dataset visually correspond to the major variations of the human body. With angle-axis, this is not guaranteed since the first modes can exhibit visually minor or major variations of the human body.

For these reasons, we exclusively use the compressed positional parameterisation for whole-body matching. Next, we discuss what changes to the matching process are required to accommodate the dimensionality reduction.

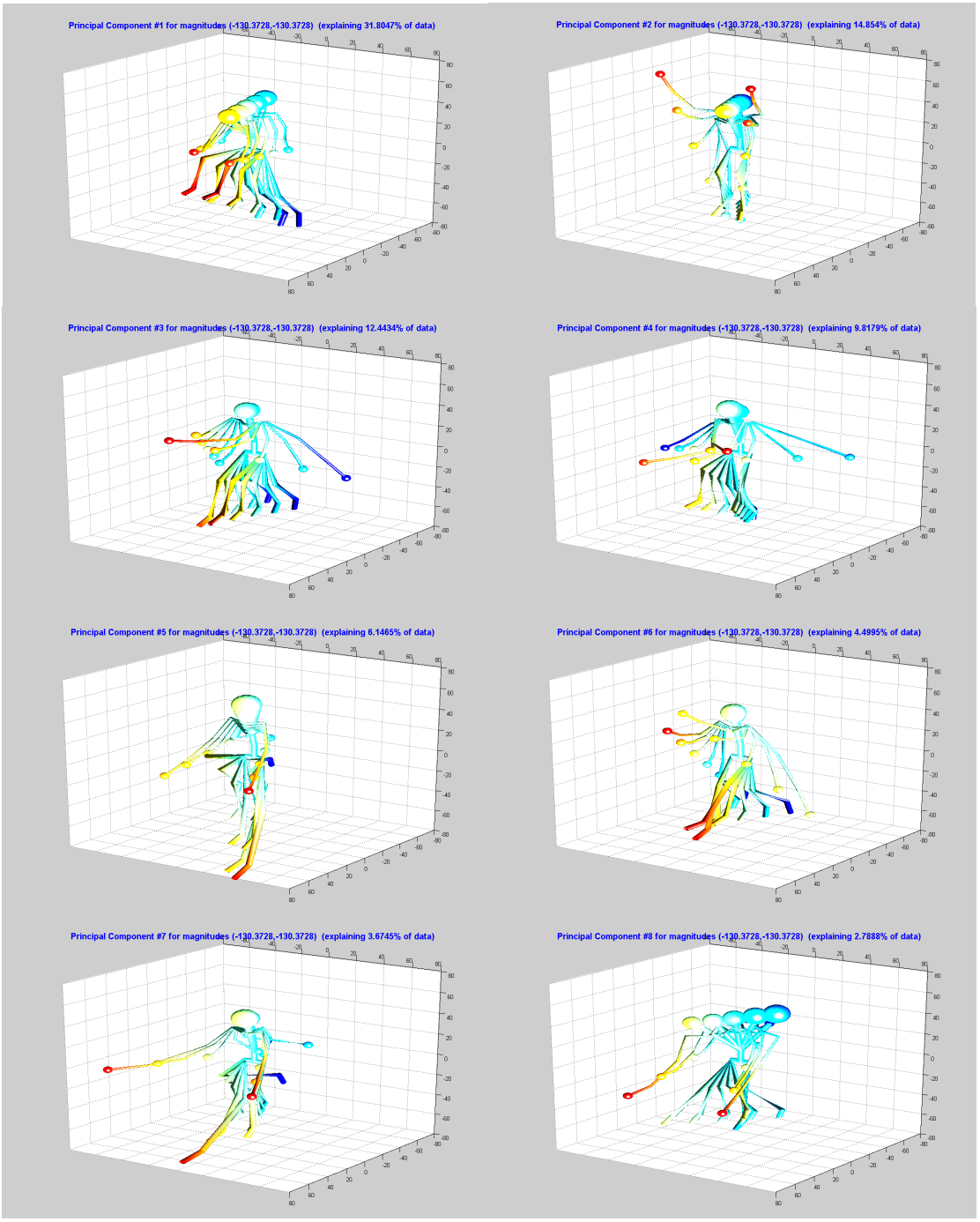

Figure 62: **Modes of the mocap dataset**. Each plot represents the range of body configurations covered by each principal component (PC). The first and most influential PC is featured at the top left and the lowest 9th PC at the bottom right. The range is depicted by plotting 5 poses linearly spread out across the minimum and maximum encountered values for that PC. Some poses might seem individually infeasible at first, but these are combined with other modes to form the correct pose. Note that global rotation and translation are ignored in this example.

3.2.7.3 Matching with Principal Component Analysis

In order to use the compressed positional representation for matching, we must ensure that distances between low-dimensional keyframes lower-bound the true distance between the raw positional data. Otherwise, the distance estimates are erroneous and the no-false-negative property is violated. Because PCA, like DFT and DWT, is an orthonormal transform, the lower-bounding property holds [Agrawal et al. 1993]. The result is that we use a (pre-selected) subset of the PCs at each key-frame as the basis for our index.

Before creating the index, PCA is executed over the whole mocap database producing a set of PCs and a set of coordinates in the PC space for each key-frame. We only keep a subset that represents the maximum variance of the data in the minimum number of dimensions. The index generation phase is exactly the same as that described in Section 3.2.5 except that the PC coefficient data is the base data. In turn, a more compact and efficient index is produced since the dimensionalities of the ensuing MBRs are reduced. Although accelerated, the candidate pruning process is mostly unchanged: the estimates are extracted from the PCA index and the true distances between sequences still require retrieval of the complete motions from disk. The number of PCs retained for the index directly impacts the accuracy of the estimate. Fewer dimensions will produce looser estimations of true distances with repercussions on the pruning power. So a trade-off between the index dimensionality and the estimate tightness is necessary. We provide experimental data in Section 4.1.2.

We are concerned with full-body matching so the body matching area is fixed. The user can select which PC dimensions to match over; but there is little incentive since reconstructed poses (i.e. eigenposes), as depicted in Figure 62, are not necessarily meaningful to an animator. The same is true for the wDTW and Euclidean distance weights where each eigenpose dimension's impact in the final distance is configurable. Depending on the application area, a PCA index is used for fast full-body matching and a rotational-based index is used when the matching area is dynamic.

3.3 User-Control of Matching

In this section, we explain how the animator interacts with the matching algorithm. First, we look at how body areas are weighted in Section 3.3.1. An optimisation that takes advantage of the inherent structure and symmetry of mocap data is presented in Section 3.3.2. Secondly, we look at how the user sorts through the returned matches. We therefore address the issue of trivial match avoidance in Section 3.3.3 and propose a simplified match selection scheme in Section 3.3.4.

3.3.1 Matching Weights and Thresholds

The weighting scheme is as simple or elaborate as the user wishes. For the Euclidean and wDTW measures, we preset the system to use Wang and Bodenheimer's [2003] predefined perceptually-motivated weights for each joint. For LCSS, we found that setting the spatial threshold for each dimension relative to a percentage (usually 10% to 30%) of its standard deviation worked well.

The ability to interactively select the body areas utilized in the matching is accomplished by multiplying the existing weights by a user-specified value. Unwanted joint motions can be defined by applying negative multipliers to the weights (for wDTW) or thresholds (for LCSS). This is shown in Figure 63. Finally, on all three measures, the user is given the possibility to search for matches that are of a certain percentage shorter or longer than the query with the uniform scaling feature (see Section 3.2.4).

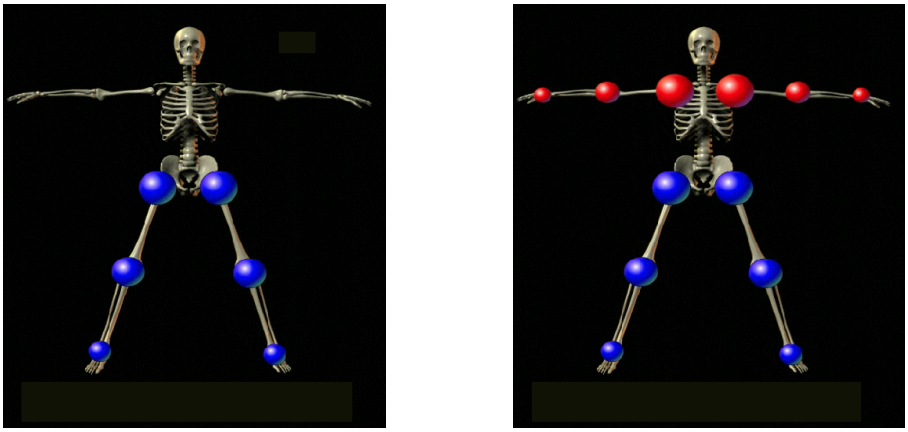

Figure 63**: Body Area Weighting**. (*Left*) The user can limit the matching to the legs only by assigning weights to individual joints proportional to their importance. (*Right*) Negative matching areas can be defined so that only motions dissimilar to the query's arm motion, and similar in leg motion, are returned.

3.3.2 Query Mirroring

To simplify the query definition process, our system supports query angle mirroring by making use of the fact that the human body is symmetric. Queries with a left-handed punch would only return motion sequences with similar left-handed punches. The animator might want to find punching motions independent of the arm performing the motion. By issuing the query twice, once in its original form and once with inverted angles, we can find all punching motions with either arm (Figure 64). Not all angles need to be inverted to form the mirror of the query. Hence, the set of invertible angles is manually pre-defined as it is dependant on the configuration of the utilized skeleton.

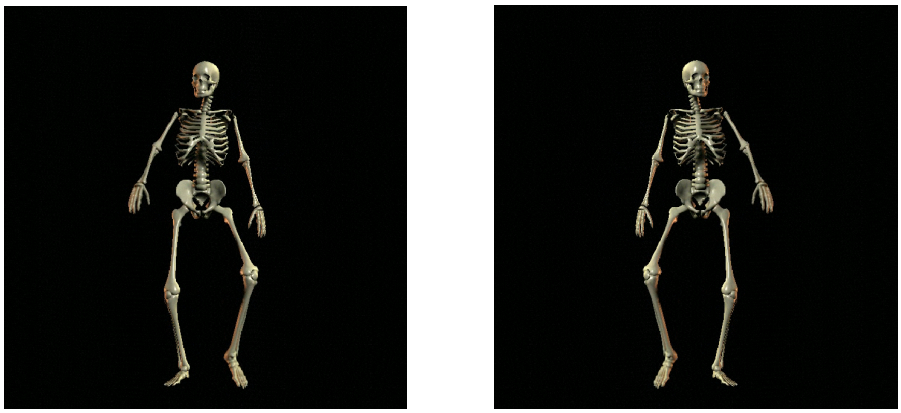

Figure 64: **Query Mirroring**. (*Left*) A frame from the original query. (*Right*) The same from the mirrored query.

3.3.3 Avoiding Trivial Matches

The best matches to any subsequence tend to be just slightly shifted versions of the subsequence. Figure 65 illustrates the idea. The result set for a *k*NN query will tend to be populated with trivial matches, limiting the value of matches beyond the first. A more appropriate match selection scheme is to enforce a heuristic minimum distance rule which prunes all overlapping and close matches. Adjacent matches are retained if their minimum inter-onset interval is above $m \cdot |Q|$, where *m* is typically set to 0.8. The process is illustrated in Figure 66. This is equivalent to scanning the distance function for local minima (with the addition that overlapping minima are also ignored). If required, the animator can regulate the minimum inter-onset interval to any value at query time.

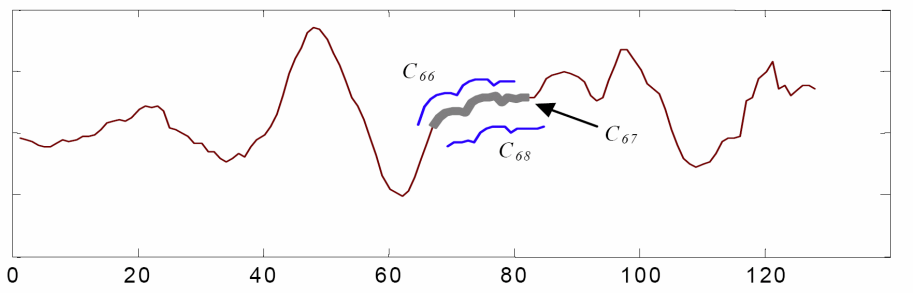

Figure 65: **Trivial Matches**. For almost any subsequence in a time series, the closest matching subsequences are the subsequences immediately to the left and right. Here, the best match is located at offset frame 67 of candidate sequence *C*. The closest best matches are located at the previous offset 66 and the next one at 68.

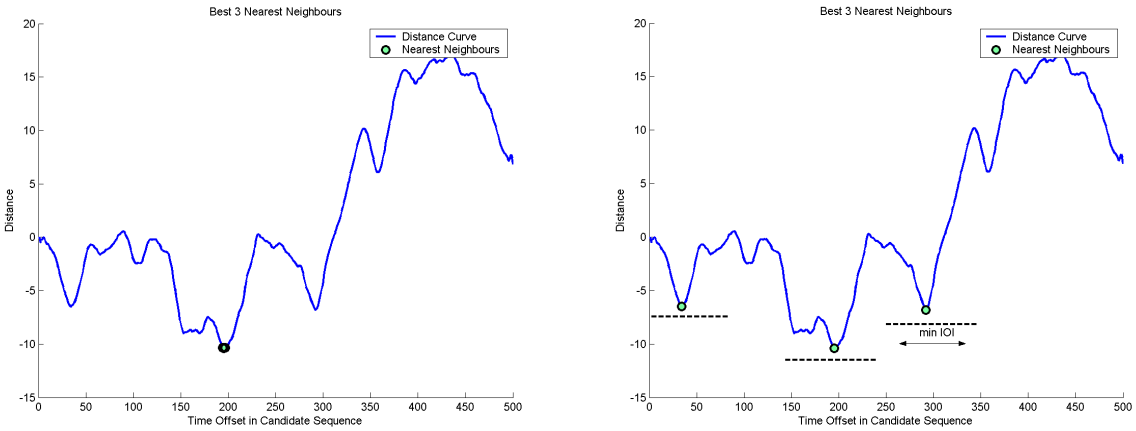

Figure 66: **Enforcing a minimum inter-onset interval**. (*Left*) The plot shows an example of a distance function computed over all 500 subsequence offsets of the candidate sequence. The three nearest neighbours, shown by green and black circles, overlap at the global distance minima. (*Right*) By enforcing a minimum distance between valid matches (in dotted), a more meaningful set of matches is returned.

3.3.4 Clustering Matches

For larger mocap databases, the set of matches can become very large. Since only a limited number of matches can be displayed at once, the query results are grouped using hierarchical clustering [Keogh and Pazzani 1999] to identify the representative centre of each cluster (Figure 67). Classification requires that a distance function between matches is defined. We use our weighted Euclidean measure (Section 3.2.3.2) over the whole body with preset weights [Wang and Bodenheimer 2003] for this purpose. The rationale is that each medoid (i.e. cluster centre) motion will effectively exemplify the general properties of its cluster members. For example, if matching is carried out on the upper-body, there may be a great variety in leg motions in the matches. The intention is to cluster stationary, running, walking and kicking motion

sequences into separate clusters. The number of clusters is proportional to a pre-defined percentage of the total number of returned matches, and is clamped for very large numbers of matches [Kaufman and Rousseeuw 1990]. The medoid motions can be simultaneously displayed so that the animator can click on each medoid to view other member motions in that cluster. Finally, unwanted medoids can be deselected by the animator so as to rapidly dismiss undesirable classes of matches.

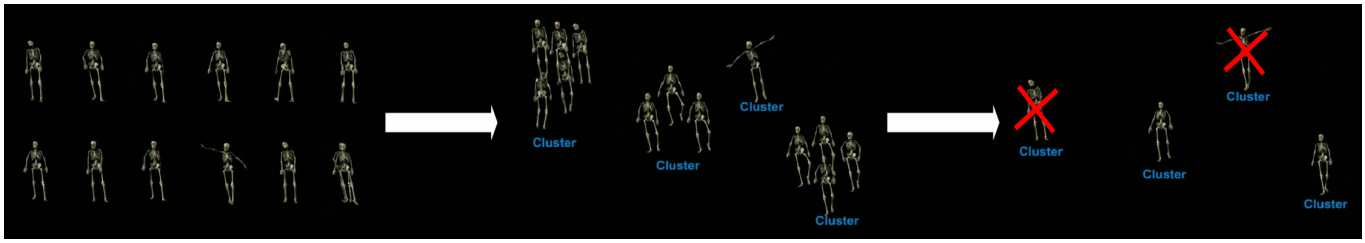

Figure 67: **Clustering Matches**. (*Left*) The original user-query returns too many matches to be displayed simultaneously. (*Middle*) The matches are clustered by their similarity. (*Right*) Only the representative cluster centres are displayed, allowing the user to quickly dismiss undesirable classes of matches.

3.4 Replicated Editing

Once acceptable matches are identified in the mocap database, replicated editing can take place. Editing operations performed on the query motion are then automatically applied repeatedly to all matches. This provides a quick and intuitive way to edit long mocap sequences or a large mocap database.

3.4.1 Applicability and Nature of Replicated Edits

Our editing system allows the animator to minimally specify alterations to a set of mocap sequences, whilst relying on the system to perform repetitive, time-consuming tasks. Applications of our replicated editing system include:

- Removing glitches due to motion capture systems.
- Editing illegal sequences inappropriate for certain skeletons or rendering conditions.
- Customizing or styling a mocap library to a particular character by adding characterisation to specific movements.
- Completely replacing recurring actions with others through replicated pasting.
- Adding variety and distinctiveness to recurrent motions.
- Eliminating offensive gestures.
- Quickly expanding a mocap database by modifying existing motions and appending the replicated edits.

Various other retouching operations are possible. For example, one could extend the system to quickly exaggerate every instance of a punch in the whole database by altering the matching sequences with the Laban *Effort* parameter [Costa et al. 2000] as well as blending in a head-nodding motion. The majority of standard motion editing techniques described in Section 2.1.2 are good candidates for replication. We now give examples of mocap editing techniques supported by our system.

Additive motion techniques [Ashraf et al. 2001; Rose et al. 1998; Kovar and Gleicher 2003] enable us to blend one or more motions into every match. We refer to this operation as replicated pasting, or cloning, since it is conceptually similar to self-similarity-based image cloning [Brooks and Dodgson 2002]. We can

also alter the range of the motion using motion warping [Bruderlin and Williams 1995], where a specifically generated displacement map is blended in without disturbing continuity and global shape in the original motion. Instead of blending in a new motion, all found matches can be entirely replaced with the query motion, or even a completely new motion. This also extends to sequence deletion. In both cases, the inserted or deleted sequence requires smooth concatenation. For deletion, this is done by stitching together [Ashraf et al. 2001; Lee and Shin 1999; Kovar et al. 2002] the gap left by the deleted sequence in such a way that the end of one motion is seamlessly connected to the start of the other.

Changes made to the mocap database during replicated editing can be promptly included in subsequent searches by simply adding their MBR sequence to the index, thereby bootstrapping the database content. This also applies to any new mocap additions made to the mocap database. Discrepancies between the search index and mocap database are consequently minimized as opposed to other indexing techniques, where a complete re-run of the pre-processing phase is usually required.

3.4.2 Contextualisation of Editing Operations

In the following section, we look at how the original edit performed on the query motion can be altered to better accommodate the current match.

In the case where the animator wants to dissimilate each detected instance of a particular motion from each other, different options are possible. Stochastic noise functions can be blended into each match in order to give each instance a distinctive feel as described in [Perlin 1995]. Similarly, the attributes of the original transformation can be randomly altered before being applied to each match. For example, the amplitude and frequency of the blended Perlin noise, or the intensity of a wave-shaping filter, can vary randomly within a given user-defined range from match to match. Our system also offers the possibility to link certain parameters of the edit to the match strength. In this manner, the more similar a match is to the query, the more intense is the applied editing operation. For example, we might want to paste a punching upper-body sequence proportionally to how close a lower-body kicking motion is to the query. We can achieve this by linking the blend intensity of the punch to the similarity between the match's and query's kicking motion.

For certain editing operations, such as displacement mapping, it is preferable to locally tailor the original transformation before applying it to each match. To this end, DTW is used to automatically establish the optimal sample correspondence between the query motion and each match. The original transform, such as the displacement map, is then remapped in time according to the calculated warp. Other edits with time-

varying attributes, such as a filtering operation with time-varying intensity or a time-warp, are also good candidates for a match-specific time remapping. This automatic alignment ensures that edits are meaningful and useful across all matches. For example, if the animator decides to amplify all punches immediately followed by a kicking motion, he would first find all occurrences of a punch followed by a kick. He would then amplify the punch in the query motion and replicate the transforms. The problem is that there might be some jitter in the start and overall duration of the punch over the set of matches. Hence, the applied amplification might not coincide perfectly with the punching motion. By re-aligning the transform, there is a greater assurance that the amplification will occur during the punching motion.

3.5 Conclusion

Capturing human motion remains a time-consuming, labour intensive and expensive process. Motion capture databases can help animators store and retrieve motions in order to re-use and create motions at a lower cost. These databases, however, require content-based retrieval methods since identifying motions with a keyword-based approach is difficult at best and requires each motion to be manually labelled. In this chapter, we have formulated the problem of content-by-retrieval over unlabelled motion capture databases and presented a novel framework for matching and editing motion capture. The problem of accurately, yet compactly, representing the human skeleton is addressed along with the non-trivial issue of evaluating similarity between two motion sequences over such skeletal representations. Keeping the system flexible is the main concern in this work. It allows the animator to define queries by multiple means and to tailor the similarity measures to a particular application area. The facility to prune trivial matches and organise matches into clusters improves the usability of the framework. Ultimately, our framework succeeds at achieving the goal of speeding up brute force matching over large databases of human motion capture. It works equally well on a series of very long or short motions.

Chapter 4

RESULTS

In his chapter, a detailed performance and quality evaluation of our matching algorithm is given. The results are assessed using several criteria: index generation and update time, index memory requirements, the query response time compared to the brute force approach and the impact of different parameter sets. The usefulness of our editing system is then illustrated by a series of practical replicated edits.

4.1 Motion Matching and Editing Results

4.1.1 Index Size and Generation Time

The index creation time scales linearly with the size of the database and takes 16:28 minutes to calculate for one hour (or 77865 keyframes) of motion (Figure 124 (left)). Supporting scale-independence prolongs the post-processing phase with costs increasing as the scale range is augmented. For example, a scale range of 20% takes 20:46 minutes for one hour of motion (Figure 124 (left)).

Increasing the number of MBRs used to represent each sequence yields tighter approximations at the cost of memory and I/O. A split ratio of 10% corresponds to $0.1 \cdot |C|$ splits per sequence. Figure 124 (right) shows the effect of the number of MBR splits on the size of the index. The index occupies 1.8MB which is less than 10% of the original database size for a split ratio of 5%. Performing PCA on one hour of mocap takes 22.8 seconds if the whole database is loaded in memory.

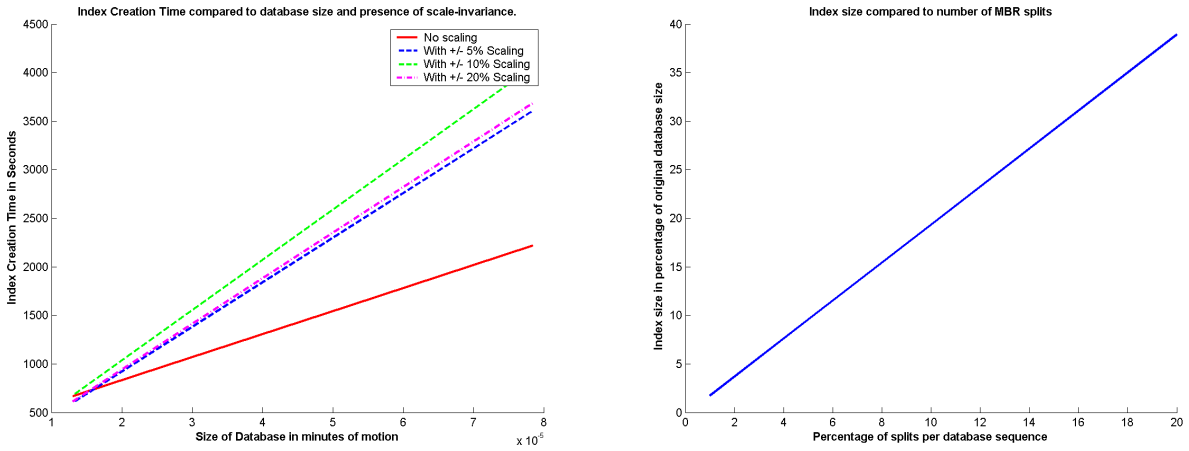

Figure 68: (*Left*) Index creation time compared to database size and presence of scale-invariance. (*Right*) Index size compared to number of MBR splits.

Any update performed on the mocap database, such as addition, deletion or modification, requires MBR recalculations only for the new or modified sequence MBRs. This is negligible during replicated editing since usually only relatively few and short sequences are affected. When a PCA compressed index is used, each frame is simply projected into the subspace defined by the pre-calculated Principal Components.

4.1.2 Performance

In this section, indexed search is compared to linear scan by analysing its normalized execution time. This is the ratio of the average time to execute a query using indexed search to the average time required to perform linear scan. The normalized cost of linear scan is 1.0. The average response time is obtained by measuring the time taken for each algorithm to process a single query and repeating the operation 20 times over randomly picked queries. We compare only with the brute force search algorithm because there are no other comparable techniques that support weightable multi-dimensional subsequence matching with scale invariance under non-metric distance measures.

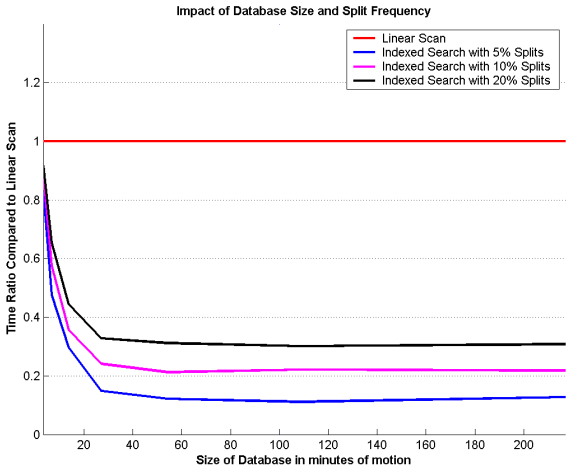

Figure 69: **Index Performance versus Database Size and Splitting Frequency**. Response times averaged over the 3 nearest neighbours of 20 random 2 second 10-dimensional queries under LCSS with temporal threshold δ=10% and spatial threshold ε=0.2 and under DTW with temporal threshold δ=10%.

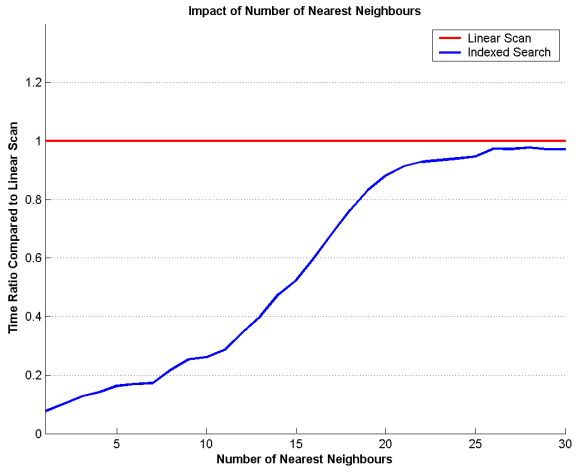

Figure 70: **Index Performance versus Number of Returned Matches.** Response times averaged over 20 random 2 second 10-dimensional queries under LCSS with temporal threshold δ=10% and spatial threshold ε=0.2 and under DTW with temporal threshold δ=10% on a one hour database.

We first turn our attention to the scalability issues and examine the behaviour of the algorithms when we vary the size of the motion capture database. Figure 69 shows that the performance of indexed search improves with increased database size as more pruning occurs, noting that the advantage tailors off after a threshold size (in this case 40 minutes of mocap). For large databases, the benefits of indexed search are considerable. For example, over a two hour database under the stated matching conditions, the search time is reduced from 5:32 minutes to 29 seconds, a speedup of up to 10 times faster compared to linear scan. The plot also demonstrates the impact of the MBR splitting frequency, which only affects estimate tightness and not final match accuracy. With split frequencies of 10% and 20%, the increase in estimate tightness is negated by the I/O costs; instead the 5% split scenario is a better compromise yielding better performance.

Figure 70 gives the behaviour of the algorithms with increasing numbers of returned matches. The number of nearest neighbours has no impact on linear scan, resulting in a constant response time, but it has significant influence on indexed search. As more nearest neighbours are included, more of the real distances are calculated from the sorted estimate list. When only the first best matches are required, two orders of magnitude speedup are achieved over the brute force approach. After more than 25 nearest neighbours, the cost of extracting the estimates and the number of visited estimates means the added benefits of the indexed search are minimal.

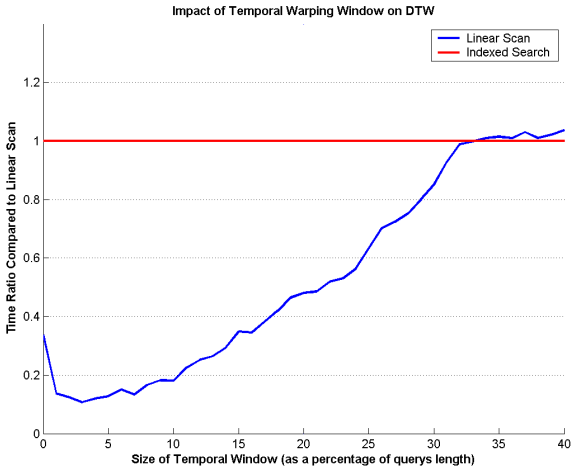

Figure 71: **Index Performance versus DTW Temporal Window**. Response times averaged over the 5 nearest neighbours of 20 random 2 second 10-dimensional queries on a one hour database.

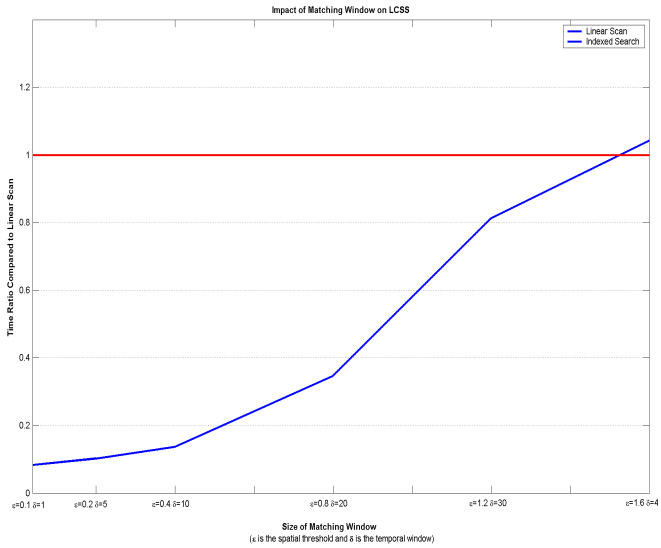

Figure 72: **Index Performance versus LCSS Temporal and Spatial Thresholds**. Response times averaged over the 3 nearest neighbours of 20 random 2 second 10-dimensional queries on a one hour database.

Indexed search performance is better with smaller DTW temporal windows, which degrades rapidly as the window size is increased in Figure 71. For large windows the MBRs enclosing the query are larger lowering the estimates closer to zero and reducing their effectiveness in separating promising matches from unpromising ones. The same is true if we increase the size of the LCSS matching envelope in Figure 72. In both cases, after a certain window/envelope size, the indexed search process becomes more costly than linear scan due to the increased search space and less accurate distance/similarity estimates.

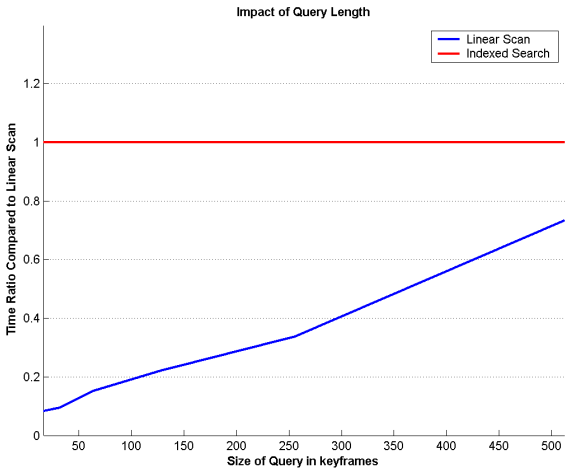

Figure 73: **Index Performance versus Query Length**. Response times averaged over the 3 nearest neighbours of 20 random 17-dimensional queries under LCSS with temporal threshold δ=10% and spatial threshold ε=0.2 and under DTW with temporal threshold δ=10% on a one hour database.

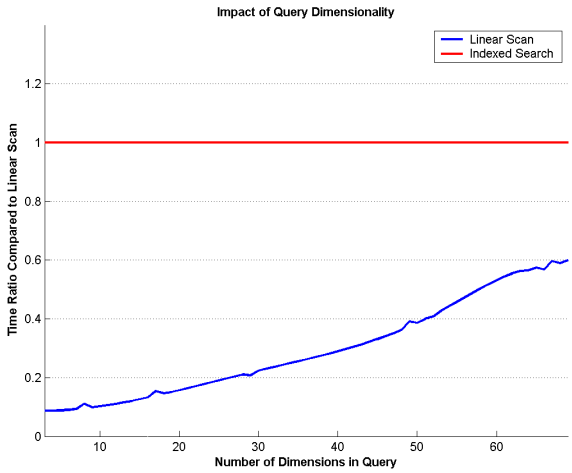

Figure 74: **Index Performance versus Query Dimensionality**. Response times averaged over the 3 nearest neighbours of 20 random 2 second queries under LCSS with temporal threshold δ=10% and spatial threshold ε=0.2 and under DTW with temporal threshold δ=10% on a one hour database.

The length of the query also has a substantial impact on performance as illustrated in Figure 73. The longer the query, the slower the speedup compared to linear scan. This is due to the volume increase of the MBRs which in turn decreases the accuracy of the similarity/distance estimates. The same is true if the dimensionality is increased (Figure 74). Fortunately, during full-body matching, its impact is mitigated by using the compressed PCA body representation which speeds up the estimation calculation phase. Under the same DTW matching conditions as in Figure 74, a speedup ranging between 80-150% is possible by using only the first 11 Principal Components (containing 94.7% of the datasets variability). We found that using fewer Principal Components yielded poorer estimates, and using more yielded minimal improvements in estimate accuracy at increased I/O costs. An added benefit of PCA is that the index requires 5-6 times less space in this compressed form.

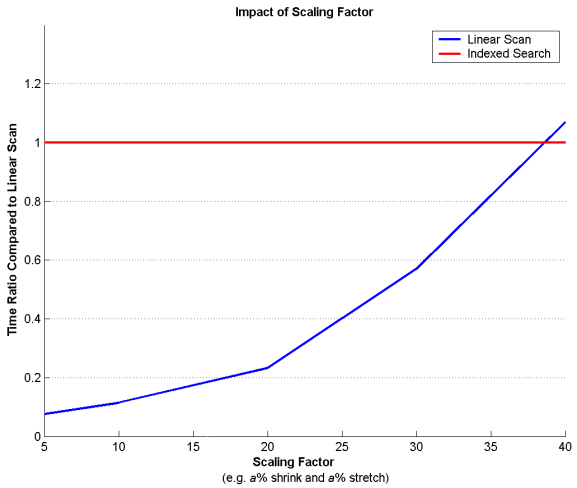

Figure 75: **Index Performance versus Scaling Factor**. The x axis indices represent a scaling factor ranging from ±5 to ±40. Response times averaged over the 2 nearest neighbours of 20 random 2 second 10-dimensional queries under LCSS with temporal threshold δ=10% and spatial threshold ε=0.2 and under DTW with temporal threshold δ=10% on a one hour database.

The above experiments do not support scale-invariance. This is considered in the experiment underlying Figure 75 which shows the effect of the scaling factor. Supporting scale-invariance under linear search is extremely costly, taking up to 2:23 hours to find the two nearest neighbours under ±10 scaling. Indexed search takes only 16.1 minutes, i.e. it is 8.7 times faster. The experimental evaluation indicates that for scaling factors ranging from 5% to 25% it is beneficial to first identify the most promising candidates entirely in the reduced dimensionality space, and then test the query against the candidate time series, despite the fact that these calculations are not as precise. Indexed search avoids the cost incurred by linear scan by postponing the expensive *TestAllScalings*() operation till the end. As expected, the larger the supported shrinking and stretching range, the less effective is indexed search.

4.1.3 Matching Quality

A critical aspect of our system is the quality of the returned matches. This depends on the rarity of the query's motion sequence, and similarly on the comprehensiveness of the dataset to find appropriate matches. The significance of matches is also dependant on the current match settings. Matches quickly become nonsensical if too lenient spatial thresholds or improper weights are utilized. This, however, is more of an attribute of our framework than an impediment since the animator is not locked into a pre-defined and inflexible similarity measure. To address this issue, we provided preset matching weights as well as the temporal and spatial thresholds. For instance, the animator is not forced to specify matching weights for each DoF. Instead, a set of pre-defined DoF weights are made available. Hence, body parts can be quickly

excluded or included without the need to redefine each individual DoF's weight. We found it practical to make minor changes to the preset weights in order to capture specific movements, e.g. to favour hand-waving motions in the returned matches. Not surprisingly, using very limited matching areas can sometimes generate counter-intuitive results due to acceptable candidates found in unanticipated motion sequences. Finally, we found that temporal windows in the range of ±10% to ±20% provided the best results. We also found that, for similar matching parameters, the quality of matches under LCSS and DTW are comparable. The advantage with LCSS is that it allows highly controllable time-varying matching parameters. In particular, the spatial range can vary independently across time and DoF dimension. The result is that the matching tolerance for any DoF of the query at any point in time is configurable. We can be sure that matches with a similarity equal to 1 are perfectly enclosed in the matching envelope. No such guarantees are possible with the less configurable DTW. However, DTW will always return matches, unlike LCSS when no motions fit inside its envelope. The DTW measure requires less initial setup since no matching ranges are enforced. If configurability and support for time-axis alignment is not an issue, then the Euclidean metric is the fastest alternative. Another disadvantage of LCSS under its current form is its inability to separate matches that are completely enclosed in the matching envelope. DTW, on the other hand, will always return a sorted list of nearest neighbours.

To test the match quality, modified versions of a query motion were planted at random locations in the database. Transformations included the addition of Perlin noise [Perlin 1995] to test tolerance to noise, random compression and decompression in time to test for temporal tolerance, and global shrinking and stretching to test for uniform scaling. Searches were then conducted with various matching parameters to verify that our system could cope effectively.

Finally, since the matching algorithm has no semantic understanding of what the motions mean, it returned matches that were visually similar but not necessarily semantically similar. For example, the beginning of a hand-shaking motion will be visually similar to the beginning of a punching motion. Only if additional meta-data semantically describing motions was incorporated in the database could this be avoided. Example matching results are available on the accompanying DVD-ROM.

4.1.4 Editing Results

On the DVD-ROM, a set of replicated edits are included that demonstrate replicated time-warping, motion warping, noise addition, emotional transform (i.e. EMOTE *Scaling* [Costa et al. 2000]) and pasting operations. Examples are given where the amplitude of the operations is both constant (that is, the same

operation is performed on all matches) or proportional to the match distance. Figure 76 shows an example of returned matches on which replicated editing is performed.

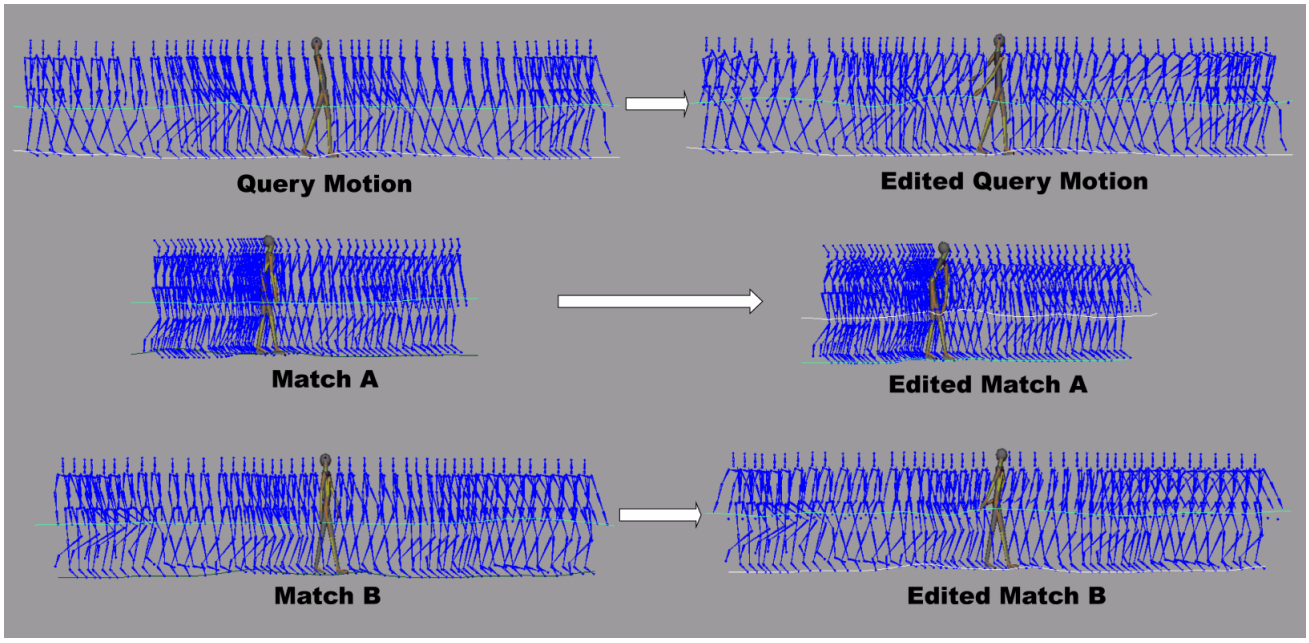

Figure 76: **Replicated editing**. The original user-query (*Top-Left*) is edited by amplifying the arm motions and adding a small offset to bend the back (*Top-Right*). The same editing operations are automatically performed onto matches A and B (*Middle and Bottom*).

In the next example, we demonstrate in more detail the use of contextualisation in the case of displacement mapping. First, a query defining a walking sequence with a rising left arm motion is submitted to the matching algorithm. A series of matches are returned. The original query sequence is edited by adding a displacement to the right arm to synchronously match the motion of the left arm. The first row of Figure 77 illustrates the alteration. Both arms are thus made to move upwards in unison. The goal is then to replicate the edit onto all matches. If we apply the original edit onto each match, the synchronicity between the left and right arm will be lost. This is because the matches will not necessarily be perfectly synchronized with the query, especially if we use DTW, as in this case. Therefore, the optimal time-warp is calculated between the query's left arm movement and that of each match. The original edit's displacement map is then resampled by the match's time-warp before being applied. The new edit accounts for the time difference between the query's and the match's left arm motions. The effect is that the replicated edit conserves the synchronicity between both arm motions across all matches, as in the edited query. The bottom row of Figure 77 show

the differences between the original edit and its contextualised counterpart. The plots for the corresponding arm motions are given in Figure 78.

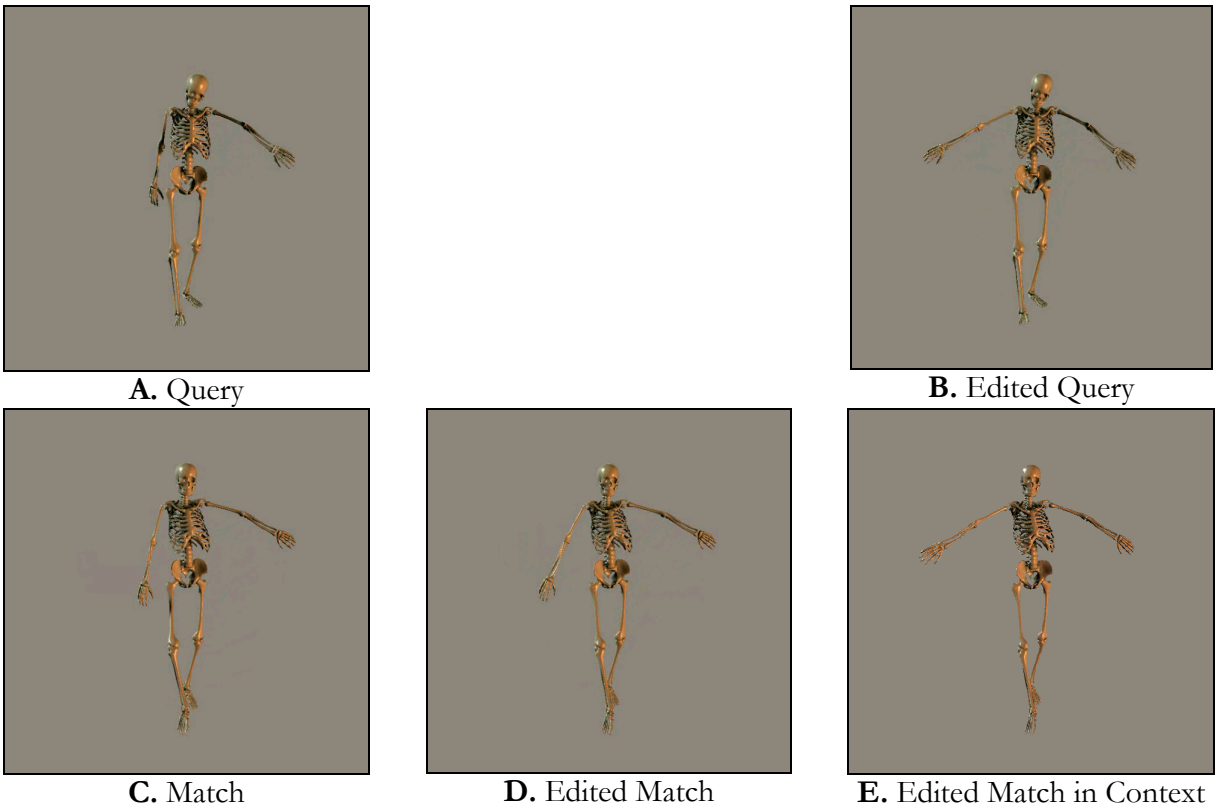

Figure 77: **Contextualised Replicated Edit**. (***A***) A frame of the query motion. (***B***) The same frame of the edited query motion where the right arm motion is made to match that of the left arm. (***C***) The best match to the query for the same frame. (***D***) If we apply the original displacement map, the synchronicity between the arms is lost. (***D***) By resampling the original displacement map by the optimal timewarp between the query and the best match, we preserve the unison upwards motion of both arms.

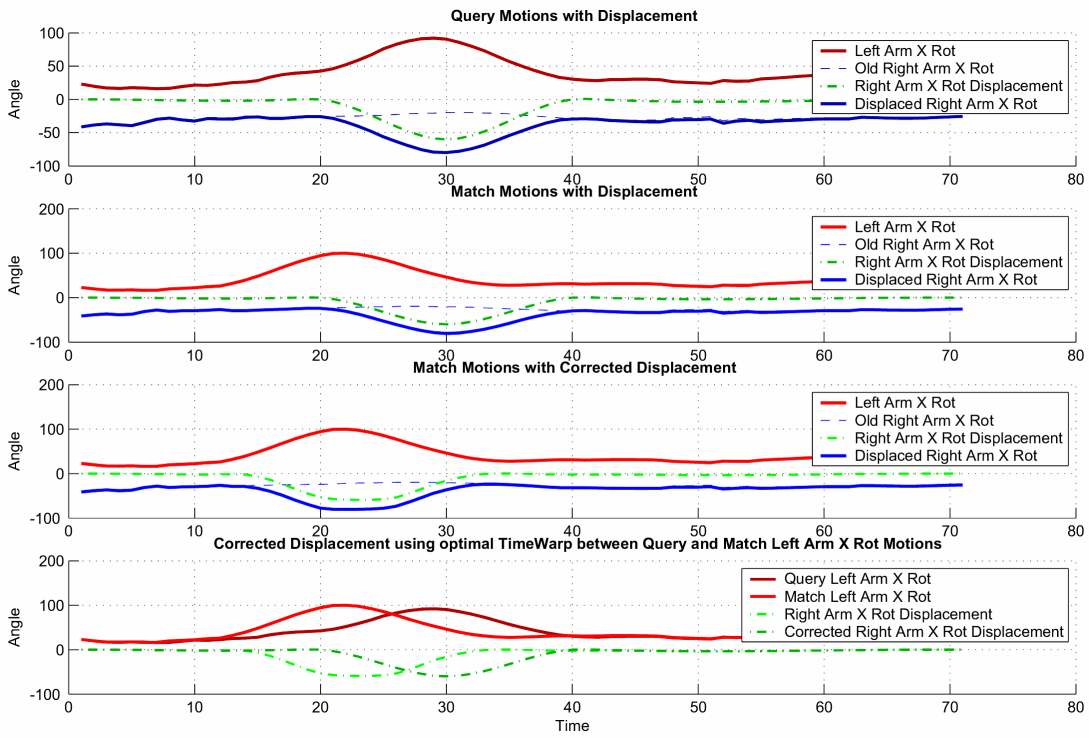

Figure 78: **Plot showing the left and right arm movements for Figure 77**. (***Top row***) Original query motion with edited right arm motion in blue. The applied displacement is in green. Notice how the right arm movements match perfectly. (***Second row***) Best match onto which the unaltered edit is applied. Notice the lack of synchronicity between the motions. (***Third row***) Same match onto which the resampled displacement is applied: now the arm motions are better synchronized. (***Last row***) The top red curves show the difference between the query and the best match's left arm motions. The bottom curves in green show the difference between the original displacement map and the resampled one, which corresponds to the difference between the upper red curves.

4.1.5 Conclusion

In this section, the performance of the matching algorithm was evaluated. It was found to offer a significant speedup for the DTW and LCSS compared to a brute force approach. The additional offline pre-processing and memory costs represent a small price to pay in comparison. Changeable matching areas, weights and distance measures allow for extensive flexibility in matching from a single index. Finally, the editing examples validate the utility and effectiveness of this approach, showing that editing operations can be quickly repeated onto matches.

Chapter 5

Conclusion and Future Work

5.1 Summary of Achievements

This work set out to simplify the production of motion for computer animation and related areas. Although only a subset of the overall issues involved in production are addressed in this report, the suggested automation techniques are nonetheless valuable as shown in our discussions.

5.1.1 Contributions to Motion Production

Animators should find our matching and replicated editing tool particularly valuable since editing motion capture is still laborious and time consuming. The benefits are more pronounced when dealing with exceedingly long mocap sequences or large mocap databases. The ability to repeat operations over many similar motions will become essential as motion capture databases expand in size.

Several contributions have been made in realising our matching and editing framework. The basic matching and editing framework was published in Cardle et al. [2003d] and scale-invariant matching in Keogh et al. [2004]. These contributions span two fields of research:

Contributions to the field of Computer Graphics

- We introduce an exact motion capture retrieval-by-content system. It is highly configurable in many respects such as the utilised similarity measure, the body matching area, the query length, the type of supported motions and the type of body representation. Unlabelled motion capture is automatically indexed in a pre-processing phase.
- The notion of replicated editing over motion data is introduced. This includes useful tools such as replicated pasting and replicated displacement. Adaptability of the applied edits is supported by allowing edit contextualisation. This tailors the edit to each match before being applied.
- Multiple similarity measures and body representations are introduced to evaluate the similarity between motions.
- A variety of query motion definitions processes are introduced such as selectable body areas on motion capture, interpolated keyframes and query mirroring.

- Two match selection methods are presented to assist sifting through extended result sets: hierarchical match clustering and trivial match elimination.

Contributions to the field of Time-series Mining

- A multi-dimensional subsequence indexed matching system with no false-negative, supporting a number of features such as rapid index update times, multiple distance measures on a single index, selectable matching dimensions, dimensional weightings, time-varying temporal and spatial thresholds and minimal inter-onset intervals between valid matches.
- Scale-invariant matching over mono-dimensional and multi-dimensional subsequences with no false-dismissal.
- The standard DTW definition is extended to the weighted multi-dimensional DTW with time-varying warping envelopes.
- The standard LCSS definition is extended to the multi-dimensional LCSS with time-varying temporal and spatial envelopes over each dimension, supporting exclusion and inclusion constraints.

5.2 Future Work

Motion production branches into many interesting sub-problems. In this section, we identify possible avenues for future research extending the capabilities of our system.

5.2.1 Motion Matching and Editing

Replicated editing can be further improved by investigating more ways of locally adapting the original edit to each motion match. In our current implementation, although possible, we found it awkward to construct queries containing motions from multiple mocap sequences. For example, the arm-waving from one sequence, the leg motion from a second and the head nodding from a third. Further research is necessary to determine the most appropriate user interface for defining single or multi-source queries, and more generally to visually organise returned matches and provide intuitive editing controls.

Our matching algorithm is able to accommodate highly configurable distance measures. It would be interesting to automatically capture the animator's subjective notion of similarity, instead of using preset weights or having the animator modify them manually. Through relevance feedback [Keogh and Pazzani 1999], the matching algorithm could refine the metric's sensitivity by automatically changing its weights and thresholds. These matching profiles can be learned continuously through interaction with the animator. Furthermore, automated techniques to find surprising, and ultimately more salient patterns [Keogh et al. 2002] in the motion data as well as automated motion annotation techniques [Arikan et al. 2003] could be used to help find more relevant matches, or to quickly eliminate whole classes of potential matches.

All sequences in our database conveniently come with identical base skeleton configurations. We do not consider the case of heterogeneous motion capture databases consisting of multiple mocap file-formats with varying base skeleton configurations. Commercial conversion utilities, such as the Dominatrix plug-in for Maya [House of Moves 2002] and MotionBuilder [Kaydara 2003], or recent work by Morales [2001] can convert between standard formats. Still, DoF correspondence and magnitude variations can make direct comparisons misleading, consequently diminishing matching accuracy and meaningfulness. To deal with sequences with greatly differing body proportions or number of DoFs, motion retargeting techniques [Gleicher 1998; Choi and Ko 1999; Lee and Shin 1999; Monzani et al. 2000] can adapt all motions to a pre-

defined reference skeleton while keeping the essence of the original motions intact (Figure 79). We leave this pre-processing phase as future work.

Figure 79: **Motion Retargeting**. The motion captured dancing sequence is slowly adapted to a smaller character during the spinning motion. The footplants and hand positions are maintained throughout using Gleicher's [1998] constraint-based optimisation.

Our system supports queries defined by a sparse set of keyframes which are then interpolated. Two techniques can be used to speed up the keyframing definition process by automatically enhancing these sketched motions. The first possibility is to use a physically-based rapid prototyping method [Lui and Popovic 2002]. Given a rough sketch of the target motion, the system infers environmental constraints and generates a realistic motion from it (Figure 80). The second approach is to use a motion texturing algorithm [Pullen and Bregler 2002], which works in a manner similar to super-resolution image synthesis [Hertzman 2001]. The approach uses motion captured data as a source for augmenting partially keyframed motions with high frequency information and additional degrees of freedom.

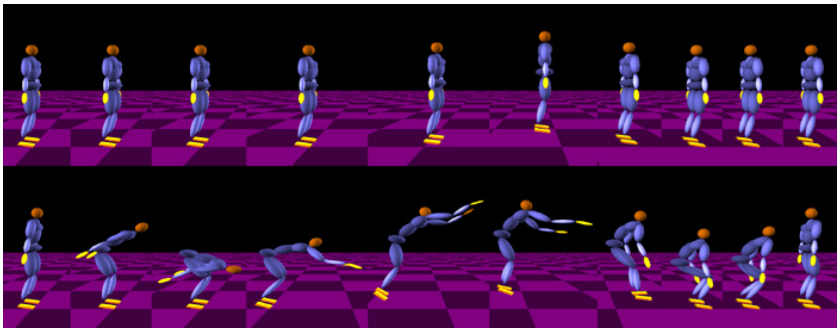

Figure 80: **Physically-based motion enhancement**. (*Top*) A simple input animation is first defined by the user, from which the algorithm synthesizes a physically realistic motion (*Bottom*)

Finally, our matching framework could be extended to match over descriptive (but still numerical) parameters automatically extracted from the motion. This would then allow the animator to find motions on a higher level than just the DoF data. Potential motion parameters include:

- ***Shape*, *Scale* and *Effort* EMOTE parameters** [Costa et al. 2000]. For example, the animator could find all motions with a similar *Effort* level to that of the query.
- **Foot Contacts**. These would be automatically extracted by detecting character footplants [Kovar et al. 2002], resulting in a binary state for each foot identifying whether it is planted or not. The animator could then find motions with footplants and arm movements (using conventional matching over the arm's DoF angle data) similar to that of the query.
- **Energy**. The energy of a motion can be defined as the sum of the absolute velocities at each joint. We could then find motions with a similar energy profile to the query.

5.2.2 Recent Work

Following our preliminary results in [Cardle et al. 2003], [Kovar and Gleicher 2004] and [Liu et al. 2003] recently introduced two mocap matching schemes. Liu et al. [2003] hierarchically cluster a motion database, where the root cluster corresponds to the hips and the leaf clusters to the end-effectors. Dimensionality is further reduced by limiting the search to automatically extracted *key-postures*. Matches are found by calculating the weighted Euclidean distance of the joints angles on all motions contained in the closest leaf cluster to the query. There is no support for subsequence matching and matching is fixed to the whole body. There is also no clear basis for selecting keyframes defining *key-postures* which can result in false-dismissal of potential clusters. With false-dismissals, the query answers do not include some motions in the database that are actually qualified answers. Kovar and Gleicher [2004] allow interactive subsequence matching over a mocap database by searching a pre-calculated time-correspondence matrix of every potential motion segment in the database to every other. *Logically* similar motions to the query are returned using a multi-step approach where all matches within a given distance threshold are used iteratively as new queries in order to find more distant motions within that same threshold. Unlike our algorithm, limited flexibility is supported so that, at query time, the matching area is fixed to the full-body with fixed joint weightings using a fixed distance function with a fixed temporal alignment window. To reduce the search

space, the sparsity of the time-correspondence matrix is reduced by estimating and pruning unpromising matches. This results in fast query response times yet introduces false-dismissals, sub-optimal time alignment and an inbuilt and fixed similarity threshold. The index is an order of magnitude larger than the original dataset and cannot support global scale invariance. We added the multi-step search functionality suggested by the authors to our system in order to find *logically* similar matches. However, in addition to degraded performance, match accuracy was adversely affected in the majority of our test cases. Ultimately, its applicability is limited since only query motions originally present in the database are supported.

ACKNOWLEDGEMENT

This work was supported by the Engineering and Physical Sciences Research Council. I would like to thank Eamonn Keogh and Michalis Vlachos for their valuable help, insights and allowing me to include some of their diagrams. I would also like to thank Stephen Brooks, Carsten Moenning, Mark Grundland, Neil Dodgson, Andras Belokosztolszki, Luis Molina-Tanco, Loic Barthe, Stephen Rymill, Marco Gillies and Tony Polichroniadis.

BIBLIOGRAPHY

Aach J. and Church G. (2001) Aligning gene expression time series with time warping algorithms. *Bioinformatics*. Volume 17, pp 495-508.

Alexa M. (2002) Linear combination of transformations. *In Proceedings of ACM SIGGRAPH 2002*, San Antonio, Texas, August 22-27.

Aggarwal C. (2001) On the Effects of Dimensionality Reduction on High Dimensional Similarity Search. *ACM PODS Conference*, 2001.

Agrawal R., Faloutsos C. and Swami A. (1993). Efficient similarity search in sequence databases. *In proceedings of the 4th Int'l Conference on Foundations of Data Organization and Algorithms*. Chicago, IL, Oct 13-15. pp 69-84.

Agrawal R., Lin K. I., Sawhney H. S., and Shim K. (1995). Fast similarity search in the presence of noise, scaling, and translation in times-series databases. *In Proc. 21st Int. Conf. on Very Large Databases*, pp. 490-501.

Argyros T and Ermopoulos C. (2003). Efficient subsequence matching in time series databases under time and amplitude transformations. *IEEE ICDM* pp 481-484.

Amaya K., Bruderlin A., and Calvert T. (1996) Emotion from motion. *In Graphics Interface '96* (May 1996), W. A. Davis and R. Bartels, Eds., pp. 222--229.

Arikan O., and Forsyth D.A. (2002). Interactive Motion Generation From Examples. *In Proceedings of ACM SIGGRAPH 2002*, San Antonio, Texas, August 22-27.

Arikan O., Forsyth D.A and O'Brien J. (2003). Motion Synthesis from Annotations. *In Proceedings of ACM SIGGRAPH 2003*, San Diego, California, August.

Ashikhmin M. (2001) Synthesizing Natural Textures. *ACM Symposium on Interactive 3D Graphics*, pp. 217–226, 2001.

Ashraf G. and Wong K. C. (2000) Generating consistent motion transition via decoupled framespace interpolation. *Computer Graphics Forum*, 2000.

Barr A. H., Currin B., Gabriel S., and Hughes J. F. (1992). Smooth interpolation of orientations with angular velocity constraints using quaternions. *Computer Graphics (Proceedings of SIGGRAPH 92)* 26, 2 (July), 313–320. ISBN 0-201-51585-7. Held in Chicago, Illinois.

Baumgartner, H. (1999) How to Catch a Ghost. *Mechanical Engineering*, Vol. 121, p108, April 99

Berndt D. and Clifford J. (1994) Using Dynamic Time Warping to Find Patterns in Time Series. *In Proc. of KDD Workshop*, 1994.

Bevilacqua F., Ridenour J., and Cuccia D. (2002) 3D motion capture data: motion analysis and mapping to music. *Proceedings of the Workshop/Symposium on Sensing and Input for Media-centric Systems 2002*, Santa Barbara, California, USA

Bowden R. (2000). Learning statistical models of human motion. *In IEEE Workshop on Human Modelling*, Analysis and Synthesis, CVPR2000.

Bowden R. and Sarhadi M., (2000) Building Temporal Models for Gesture Recognition, *Proc. of the British Machine Vision Conference 2000,* Vol 1, University of Bristol, UK, September 2000.

Boyd A. (2002) Lord of The Rings: The Two Towers. Stormfront Studios Interview. *music4games.net*. 2002.

Bozkaya T., Yazdani and Özsoyoglu Meral N. (1997) Matching and Indexing Sequences of Different Lengths. *CIKM 1997:* 128-135

BoxOfficeMojo.com (2004) http://www.boxofficemojo.com/alltime/world/. Visited in 2004.

Brand M. and Hertzmann A. (2000) Style machines. *In The Proc. of ACM SIGGRAPH 2000*, pages 183--192, 2000.

Bregler C. and Malik J. (1998) Tracking People with Twists and Exponential Maps, *CVPR 1998* (8-15).

Brooks S. and Dodgson N. (2002). Self-Similarity Based Texture Editing. *ACM SIGGRAPH 2002*.

Bruderlin A. and Williams L. (1995) Motion signal processing. *In Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, pages 97--104, August 1995.

Buss S. and Fillmore J. Spherical Averages and Applications to Spherical Splines and Interpolation. *ACM Transactions on Graphics 20* (2001) 95-126.

Cardle M., Barthe L., Brooks S. and Robinson P. (2002a) Local Motion Transformations Guided by Music Analysis. *Eurographics UK 2002*, Leicester, June 2002.

Cardle M., Vlachos M., Brooks S., Keogh E. and Gunopulos D. (2003d) Fast Motion Capture Matching with Replicated Motion Editing. *SIGGRAPH 2003* Sketches and Applications, San Diego, July 2003.

Chan K. and Fu A. W. (1999). Efficient time series matching by wavelets. *In proceedings of the 15th IEEE Int'l Conference on Data Engineering*. Sydney, Australia, Mar 23-26. pp 126-133.

Choi K-J. and Ko H.-S., (2000) Online Motion Retargetting. *The Journal of Visualization and Computer Animation* 11(5):223-235, 2000, A special issue paper selected from Pacific Graphics '99.

Chu S., Keogh E., Hart D. and Pazzani M (2002). Iterative deepening dynamic time warping for time series. *In Proc 2nd SIAM International Conference on Data Mining*.

Chu K., Lam S. and Wong, M. (1998). An efficient hash-based algorithm for sequence data searching. *The Computer Journal* 41 (6): 402-415.

Clote P., Straubhaar J. and Ewell (2003). An Application of Time Warping to Functional Genomics. Technical report. New York University Computer Science Department.

Costa M., Zhao L., Chi D., and Badler N. (2000) The EMOTE model for effort and shape. *In Proceedings of SIGGRAPH 2000*, 2000

Dam E., Koch M. and Lillholm M. (1998) Quaternions, Interpolation, and Animation. *Technical Report* DIKU-TR-98/5. University of Copenhagen

Das G., Gunopulos D. and Mannila H. (1997) Finding Similar Time Series. *In Proc. of the First PKDD Symp.*, pages 88-100, 1997.

Darwin 3D, LLC. (1998). http://www.darwin3d.com/gdm1998.htm. Visited in 2002.

Davis J. and Shah M., (1999) Toward 3-D Gesture Recognition, *Int'l Journal of Pattern Recognition and Artificial Intelligence*, Vol. 13, No. 3, May 1999.

Faloutsos C., Ranganathan M. and Manolopoulos Y. (1994). Fast subsequence matching in time-series databases. *In Proc. ACM SIGMOD Conf.*, Minneapolis.

Fang A. and Pollard N. (2003). Efficient Synthesis of Physically Valid Human Motion, *ACM Transactions on Graphics* 22(3) 417-426, SIGGRAPH 2003 Proceedings.

Gavrila D. M. and Davis L. S. (1995). Towards 3-d model-based tracking and recognition of human movement: a multi-view approach. *In International Workshop on Automatic Face- and Gesture-Recognition*.

Grassia F. S. (1998). Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools* 3, 3, 29–48. ISSN 1086-7651.

Gleicher. M (1997) Motion editing with spacetime constraints. In Michael Cohen and David Zeltzer, editors, 1997 *Symposium on Interactive 3D Graphics*, pages 139–148. ACM SIGGRAPH, April 1997. ISBN 0-89791-884-3.

Gleicher M. (1998) Retargeting motion to new characters. *In SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 33–42. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

Gleicher M. (1998) Retargeting motion to new characters. *In Computer Graphics (SIGGRAPH 98 Proceedings)*, pages 33–42, July 1998.

Gleicher. M. (2001) Motion path editing. *In Proceedings 2001 ACM Symposium on Interactive 3D Graphics*. ACM, March 2001.

Gleicher M. (2001). Comparing constraint-based motion editing methods. *Graphical Models* 63, 2, 107–123.

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

Grünvogel S. M., Piesk J., Schwichtenberg S. and Büchel G. (2002) AMOBA: A Database System for Annotating Captured Human Movements . *Proceedings of Computer Animation 2002* (CA2002), June 19-23, 2002, Geneva, Switzerland IEEE Computer Society, Los Alamitos, pp. 98 - 102

Guttman A. (1984). R-trees: A Dynamic Index Structure for Spatial Searching. *In Proceedings of ACM SIGMOD International Conf. on Management of Data*.

Hadjieleftheriou M., G. Kollios, V. Tsotras, and D. Gunopulos. (2002) Efficient indexing of spatiotemporal objects. *In Proc. of 8th EDBT*, 2002.

Hetland M. (2003). A survey of recent methods for efficient retrieval of similar time sequences. *Data Mining in Time Series Databases, World Scientific*.

Hertzman A., Javobs C., Oliver N., Curless B., and Salesin D. H. (2001). Image Analogies. *In Proceedings of SIGGRAPH 2001, Computer Graphics Proceedings*, Annual Conference Series, 327-340.

Hjaltason G. R. and Samet H. (1995) Ranking in spatial databases. *In Proceedings of the 14th Symposium on Spatial Databases*, pages 83–95, Portland, Maine, August 1995.

Home Theater Buyer's Guide Magazine, (1998) David Lynch Interview. Fall 1998.

House of Moves (2002). www.moves.com. Visited in 2002.

Itakura F. (1975). Minimum prediction residual principle applied to speech recognition. *IEEE Trans. Acoustics, Speech, and Signal Proc.*, Vol. ASSP-23, pp. 52-72

Jollife I.T. (1986) Principal Component Analysis. Springer-Verlag, New York, 1986.

Kahveci T. and Singh, A (2001). Variable length queries for time series data. *Proceedings 17 th International Conference on Data Engineering.* Heidelberg, Germany.

Kahveci T., Singh A., Gürel A. (2002) An Efficient Index Structure for Shift and Scale Invariant Search of Multi-Attribute Time Sequences. *ICDE 2002*: 266.

Kathleen R. (1997). Computer-Human Collaboration in the Design of Graphics. *PhD thesis*, Harvard University.

Kaufman L. and Rousseeuw P. (1990). Finding Groups in Data: An Introduction to Cluster Analysis, John Wiley & Sons, New York.

Kaydara Inc. (2003). http://www.kaydara.com/. Visited in 2003.

Keogh E. (2002). Exact indexing of dynamic time warping. *In 28th International Conference on Very Large Data Bases.* Hong Kong. pp 406-417.

Keogh E,. Chakrabarti K, Pazzani M. and Mehrotra (2000) Dimensionality reduction for fast similarity search in large time series databases. *Journal of Knowledge and Information Systems.*

Keogh E. and Kasetty S. (2002). On the need for time series data mining benchmarks: a survey and empirical demonstration. *In the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* Edmonton, Canada. pp 102-111.

Keogh E., Lin J., and Truppel W. (2003). Clustering of Time Series Subsequences is Meaningless: Implications for Past and Future Research. *In proceedings of the 3rd IEEE International Conference on Data Mining* . Melbourne, FL. Nov 19-22. pp 115-122.

Keogh E., Lonardi S and Chiu W. (2002). Finding Surprising Patterns in a Time Series Database In Linear Time and Space. *In the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* July 23 - 26, 2002. Edmonton, Alberta, Canada. pp 550-556.

Keogh E. and Pazzani M. (1999). Relevance feedback retrieval of time series data. *In Proceedings of the 22th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieva*l. pp 183-190.

Keogh E. and Pazzani M. (2000a). A simple dimensionality reduction technique for fast similarity search in large time series databases. *In Proceedings of Pacific- Asia Conf. on Knowledge Discovery and Data Mining*, pp 122-133.

Keogh E., and Pazzani M. (2000b). Scaling up dynamic time warping for data mining applications. *In 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* Boston.

Keogh E., Zordan V., Gunopulos D. and Cardle M. (2004) Indexing Large Human Motion Databases. *Submitted to Very Large DataBases (VLDB) 2004*, Toronto, September 2004.

Kim M.-J., Shin S.-Y. and Myung-Soo K. (1995) A General Construction Scheme for Unit Quaternion Curves With Simple High Order Derivatives, *Proceedings of SIGGRAPH 95, Computer Graphics Proceedings*, Annual Conference Series, pp. 369-376 (August 1995, Los Angeles, California). Addison Wesley. Edited by Robert Cook. ISBN 0-201-84776-0.

Kim S., Park S., and Chu W. (2001) An index-based approach for similarity search supporting time warping in large sequence databases. In *In Proc. of 17th ICDE*, 2001.

Korn F., Jagadish H. and Faloutsos C. (1997). Efficiently supporting ad hoc queries in large datasets of time sequences. *In proceedings of the ACM SIGMOD Int'l Conference on Management of Data.* Tucson, AZ, May 13-15. pp 289-300.

Kovar L., Gleicher M., and Pighin F. (2002) Motion graphs. *In Proceedings of ACM SIGGRAPH 2002, Annual Conference Series. ACM SIGGRAPH*, July 2002.

Kovar L. and Gleicher M. (2004) Automated Extraction and Parameterization of Motions in Large Datasets. *In Proceedings of ACM SIGGRAPH 2004. Los Angeles.*

Kovar L., Schreiner J., and Gleicher M. (2002) Footskate cleanup for motion capture editing. *In ACM Symposium on Computer Animation 2002*

Kovar L. and Gleicher M. (2003) Flexible Automatic Motion Blending with Registration Curves. *Eurographics/SIGGRAPH Symposium on Computer Animation (2003)*

Kruskall J. B. and Liberman M. (1983). The symmetric time warping algorithm: From continuous to discrete. In Time Warps, String Edits and Macromolecules. Addison-Wesley.

Lasseter J. (1987) Principles of Traditional Animation Applied to 3D Computer Graphics, *SIGGRAPH'87*, 1987

Lee J., Chai J., Reitsma P., Hodgins J., and Pollard N. (2002) Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics, Proceedings of ACM SIGGRAPH 2002*, 21(3):491.500, July 2002.

Lee S.-L., Chun S.-J., Kim D.-H., Lee J.-H. and Chung C.-W. (2000) Similarity Search for Multidimensional Data Sequences. In *Proceedings of the IEEE Sixteenth International Conference on Data Engineering*, pages 599--608, 2000

Lee J. and Shin S.-Y. (1999) A hierarchical approach to interactive motion editing for human-like figures. *Proceedings of SIGGRAPH 99*, pages 39–48, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

Lee J. and Shin S.-Y. (2002) General Construction of Time-Domain Filters for Orientation Data, *IEEE Transactions on Visualization and Computer Graphics*, volume 8, number 2, 119-128, 2002.

Li Y., Wang T., and Shum H-Y. (2002). Motion Textures: A Two-Level Statistical Model for Character Motion Synthesis. *In Proceedings of ACM SIGGRAPH 2002*, San Antonio, Texas, August 22-27.

Litwinowicz P. (1991) Inkwell: A 2 1/2-d animation system. *SIGGRAPH 1991 Proceedings*, 25:pages 113–122, July 1991.

Lin J., Keogh E., Lonardi S. and Patel P. (2002) Finding Motifs in Time Series. *Proc. of the 2nd Workshop on Temporal Data Mining* 2002 53–68.

Liu, F., Zhuan, Y., Wu, F., and Pan, Y. (2003). 3d motion retrieval with motion index tree. Computer Vision and Image Understanding 92, 2-3, 265-284.

Liu C. K., and Popovi'c Z. (2002). Synthesis of complex dynamic character motion from simple animations. *ACM Transactions on Graphics* 21, 3 (July), 408–416.

Medin D. L., Goldstone R. L., and Gentner D. (1993) Respects for similarity. *Psychological Review*, 100(2):254–278, 1993.

Menache A. (2000) Understanding Motion Capture for Computer Animation and Video Games. 2000. Morgan Kaufman.

Meredith M. and Maddock S. (2000). Motion Capture File Formats Explained. University of Sheffield. *Technical Report.*

Moghaddam B. (1999) Principal Manifolds and Bayesian Subspaces for Visual Recognition, *Proc. Of the 7th IEEE International Conf. Computer Vision, ICCV'99*, September, 1999.

Molina-Tanco, L. and Hilton, A. (2000). Realistic synthesis of novel human movements from a database of motion capture examples. *In Workshop on Human Motion (HUMO'00)*, 137-142.

Molina-Tanco L. (2003). Human Motion Synthesis from Captured Data. *Ph.D Thesis*. University of Surrey.

Monzani J.-S., Baerlocher P., Boulic R., Thalmann D. (2000) Using an Intermediate Skeleton and Inverse Kinematics for Motion Retargeting ,*Proc. Eurographics 2000*

Moon Y.-S. Whang K.-Y., and Loh W.-K. (2001) Duality-based subsequence matching in time-series databases. *In Proc. the 17th Int'l Conf. on Data Engineering,* pages 263-272, 2001.

Moon Y.-S., Whang K.-Y., and Han W.-S. (2002) Generalmatch: A subsequence matching method in time-series databases based on generalized windows. *In SIGMOD 2002,*Madison,Wisconsin, USA, 3-6 June 2002.

Morales C. (2001) Development of an XML Web based motion capture data warehousing and translation system for collaborative animation projects (2001). *WSCG 2001 Conference Proceedings*

Mori T., Uehara K., Shimada M. (2002) Extraction of Primitive Motion and Discovery of Association Rules from Human Motion Data. Progress in *Discovery Science 2002*: 338-348

Morinaka Y., Masatoshi Y., Toshiyuki A. and Uemura S. (2001) The L-index: An Indexing Structure for Efficient Subsequence Matching in Time Sequence Databases *Fifth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2001)*

Moeslund T.B. and Granum E. (2001) A Survey of Computer Vision-Based Human Motion Capture. *Computer Vision and Image Understanding,* 2001.

Murray R M., Zexiang L. and Shankar Sastry S. (1994) A Mathematical Introduction to Robotic Manipulation. CRC Press, Boca Raton, 1994, pages 22-34,73.

Murphy G. and Medin D. (1985) The role of theories in conceptual coherence. *Psychological Review*, 92:289–316, 1985

Ng C. W. and Ranganath S. (2000) Gesture Recognition via Pose Classification, *Proc. of the Int'l Conf. Pattern Recognition ICPR'00*, Barcelona, Spain, September 2000.

O'Donnell Marty. (2002) Producing Audio for Halo. *Game Developer Conference* 2002.

Osaki R., Shimada M., Uehara K. (1999) Extraction of Primitive Motion for Human Motion Recognition. *Discovery Science* 1999: 351-352

Osaki R., Shimada M., Uehara K. (2000) A Motion Recognition Method by Using Primitive Motions. *VDB 2000:* 117-128

Kim S.-W., Park S., Chu W. (2001) An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases. *ICDE 2001:* 607-614

Keller D., and Truax B. (1998). Ecologically-based granular synthesis. *Proceedings of the International Computer Music Conference*, Ann Arbor, IL: ICMA.

Panels Session. (2002) How Does Motion Capture Affect Animation? *Siggraph 2002*. San Antonio. Texas

Park S., Chu W.W., Yoon J., and Hsu C. (2000) Efficient searches for similar subsequences of different lengths in sequence databases. *In ICDE*, San Diego, CA, February 2000.

Park S., Kim S, and Chu W. (2001). Segment-based approach for subsequence searches in sequence databases. In *Proceedings of the 16th ACM Symposium on Applied Computing*, pp. 248-252, Las Vegas, NV, USA.

Park S. I., Shin H. J., and Shin S. Y. (2002) On-line locomotion generation based on motion blending. *In Proceedings of ACM SIGGRAPH Symposium on Computer Animation 2002*. ACM SIGGRAPH, July 2002.

Perng C.-S., Wang H., Zhang S. R., Parker D. S. (2000) Landmarks: a New Model for Similarity-based Pattern Querying in Time Series Databases, *ICDE 2000.*

Pennec X. and Thirion J.P. (1997) A framework for uncertainty and validation of 3D registration methods based on points and frames. *Int. Journal of Computer Vision*, in press, 1997.

Perlin K. (1995) Real-time Responsive Animation with Personality. *In IEEE Transactions on Visualization and Computer Graphics*, 1(1), March 1995, pp.5-15.

Polichroniadis T. (2000) High Level Control of Virtual Actors. University of Cambridge. *Ph.D Thesis.*

Popovic Z. and Witkin A. (1999). Physically based motion transformation. *In Proceedings of ACM SIGGRAPH 99*, pages 11–20.

Pullen K., and Bregler C. (2002). Motion Capture Assisted Animation: Texturing and Synthesis, *In Proceedings of ACM SIGGRAPH 2002,* San Antonio, Texas, August 22-27.

Rabiner L. and Juang B. (1993). Fundamentals of speech recognition. Englewood Cliffs, N.J, Prentice Hall.

Rabiner L., Rosenberg A. and Levinson S. (1978). Considerations in dynamic time warping algorithms for discrete word recognition. *IEEE Trans. Acoustics, Speech, and Signal Proc.*, Vol. ASSP-26, pp. 575-582.

Robertson B. Bad to the Bone. (1999) *Computer Graphics World.*, Vol. 22, No. 5, p34 May 99.

Roddick J.F. and Hornsby K. (2000) Temporal, Spatial and Spatio-Temporal Data Mining. 2000.

Rose C., Guenter B., Bodenheimer B., and Cohen M. (1996) Efficient generation of motion transitions using spacetime constraints. *In Proceedings of ACM SIGGRAPH 1996*, Annual Conference Series, pages 147–154. ACM SIGGRAPH, August 1996.

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

Roussopoulos N., Kelley S., and Vincent F. (1995) Nearest neighbor queries. *In ACM SIGMOD*, San Jose, CA, 1995.

Sakoe H. and Chiba S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. Acoustics, Speech, and Signal Proc.*, Vol. ASSP-26. pp. 43-49.

Schwartz D. (2003) The Caterpillar system for data-driven concatenate sound synthesis. *Proc. of the 6th Int. Conference on Digital Audio Effects (DAFx-03)*, London, UK, September 8-11, 2003.

Schödl A., Szeliski R., Salesin and D., Essa I. (2000). Video textures. *In Proceedings of SIGGRAPH 2000*, pages 489-498, July.

Shimada M., Uehara K. (2000) Discovering of Correlation from Multi-stream of Human Motion. *Discovery Science 2000:* 290-294

Shoemake K. (1985). Animating rotation with quaternion curves. *Computer Graphics (Proceedings of SIGGRAPH 85)* 19, 3 (July), 245–254. Held in San Francisco, California.

Shoemake K. (1991). Quaternions and 4x4 matrices. *Graphics Gems II,* 351–354. ISBN 0-12-064481-9. Held in Boston.

Sidenbladh H., Black M. J., and Sigal L. (2002). Implicit probabilistic models of human motion for synthesis and tracking. *In European Conference on Computer Vision (ECCV).*

Stanford University. (2000) Graphics Course CS448. http://graphics.stanford.edu/courses/cs448a-00-fall/ 2002

Starner T. and Pentland A. (1995) Visual Recognition of American Sign Language Using Hidden Markov Models," *Proc. of the Int'l Workshop on Automatic Face and Gesture Recognition*, Zurich, June 1995

Sturman D. (1994) A Brief History of Motion Capture for Computer Character Animation, Character Motion Systems, *Siggraph 94, Course 9*.

Su M., Huang H., Lin C., Huang C., and Lin C. (1998) Application of Neural Networks in Spatio-Temporal Hand Gesture Recognition. *Proc. of the IEEE World Congress on Computational Intelligence*, USA, 1998.

Tak S, Song H-Y. and Seok Ko H. (2000). Motion Balance Filtering. *EUROGRAPHICS 2000,* Computer Graphics Forum, Vol. 19, No. 3.

Tanaka Y., Uehara K. (2003) Discover Motifs in Multi-dimensional Time-Series Using the Principal Component Analysis and the MDL Principle. *MLDM 2003:* 252-265

Tappert C. and Das S. (1978). Memory and time improvements in a dynamic programming algorithm for matching speech patterns. *IEEE Trans. Acoustics, Speech, and Signal Proc.*, Vol. ASSP-26, pp. 583-586.

Tversky A. (1977) Feature of similarity. *Psychological Review*, 84:327–352, 1977.

Unuma M., Anjyo K. and Takeuchi R. (1995) Fourier Principles for Emotion-based Human Figure Animation. *SIGGRAPH '95,* 1995.

Vlachos M., Kollios G., and Gunopulos D. (2002) Discovering similar multidimensional trajectories. *In ICDE,* pages 673–684, San Jose, CA, 2002.

Vlachos M., Hadjieleftheriou M., Gunopulos D., Keogh E. (2003) Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures, *In Proc. of 9th International Conf. on Knowledge Discovery & Data Mining (SIGKDD)*, Washington, DC, 2003

Waibel A. and Lee K.-F., editors. (1990) Readings in Speech Recognition. Morgan Kaufmann Publishers, San Mateo, CA, 1990.

Wang J., and Bodenheimer B. (2003), An Evaluation of a Cost Metric for Selecting Transitions between Motion Segments', 2003 *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 232-238, San Diego, CA, July 2003.

Weber R. (1998) A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," *In Proc. Int'l Conf. on Very Large Data Bases*, pp. 194-205, New York, New York, Aug. 1998.

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

Witkin A. and Popovic. Z. (1995) Motion warping. *In SIGGRAPH 95 Proceedings, Annual Conference Series*, pages 105--108. ACM SIGGRAPH

Yi B.-K., Jagadish H. V., Faloutsos C. (1998) Efficient Retrieval of Similar Time Sequences Under Time Warping. *ICDE 1998:* 201-208

Yi B. and Faloutsos C. (2000). Fast time sequence indexing for arbitrary lp norms. *In proceedings of the 26th Int'l Conference on Very Large Databases.* Cairo, Egypt, Sept 10-14. pp 385-394.

Yi B., Jagadish K. and Faloutsos C. (1998). Efficient retrieval of similar time sequences under time warping. *In ICDE 98*, pp. 23-27.

Zhu Y., Shasha D., and Zhao X. (2003). Query by humming: a time series database approach. *In Proc. of SIGMOD, 2003.* pp. 675.

Zils A. and Pachet F. (2001) Musical mosaicing. *In Proc. Of G-6 Conference Digital Audio Effects DAFX-01*, Limerick, Ireland, 2001.

APPENDIX A

ANGLE REPRESENTATIONS

This section covers in more details the three angle representations we considered for joint parameterization. These include Euler angles, Angle-Axis and Quaternions.

Euler

Representing a rotation with Euler angles is probably the best known and simplest parameterisation. A rotation is expressed compactly in an ordered set of three rotations typically around the pre-chosen X, Y and Z axes (Figure 81).

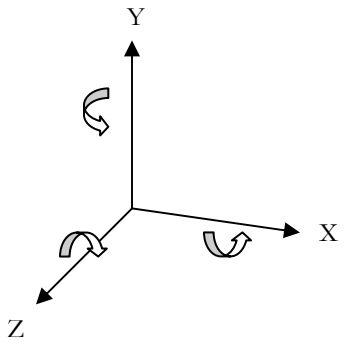

Figure 81: Rotation axes of Euler rotation.

Because Euler angles are Euclidean quantities, the distance between two rotations becomes the Euclidean distance between the corresponding angle values which is simple to bound and fast to calculate. For the same reasons, the motion can be quickly resampled using linear or cubic-spline interpolation. The Euler form of the distance metric between two character poses P_1 and P_2 is the weighted difference of the joint orientations:

$$d(P_1, P_2) = \sum_{j=1}^{J} w_j \left\| \boldsymbol{\varphi}_1^j - \boldsymbol{\varphi}_2^j \right\|^2 \qquad \text{with} \quad \sum_{j=1}^{J} w_j = 1$$

j is the current joint

J is the total number of joints in the character hierarchy.

w_j is a weighting value for each joint

$\boldsymbol{\varphi}_1^j \in \Re^3$ is the vector of 3 axial rotations R_x, R_y and R_z at the j -th joint of pose P_1

$\boldsymbol{\varphi}_2^j \in \Re^3$ is the vector of 3 axial rotations R_x, R_y and R_z at the j -th joint of pose P_2

A problem arises from the fact that Euler angles do not satisfy our singularity requirement. It suffers from *gimbal lock* singularities where one Degree of Freedom (DoF) is lost when one of the first rotation is done to 90 degrees *i.e.* two of the three Euler angles belong to the same DoF. The more general form of this problem is that for any rotation there is more than one way to represent it; and that there is not necessarily a connection between how close the numbers are and how similar the rotations are. Similarly, interpolation also suffers since each axis is interpolated independently, resulting in the loss of interdependence between axes. Converting an Euler angle to its matrix form counterpart will only increase the parameter space from 3 to 9 (or 16 in the case of homogeneous transforms), requiring computationally expensive constraint enforcements, and ultimately will suffer from the same singularities. The inability of the distance metric based on Euler angle rotations to separate distinct poses by a greater magnitude than those which appear similar leads to an unreliable inter-pose metric, consequently leading to poor matching quality.

Angle-axis

An arbitrary rotation $R \in SO(3)$, where $SO(3)$ is the group of 3D rotations in Euclidean space, can be represented by the so-called *angle-axis* representation, in which the axis of rotation $\boldsymbol{a} \in \Re^3$ and of rotation θ about $\boldsymbol{a}$ are given separately (Figure 82).

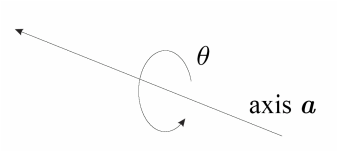

Figure 82: Rotation around axis $\mathbf{a}$ and angle θ

Since only the direction of the rotation axis $\boldsymbol{a}$ is of importance, $\boldsymbol{a}$ has only 2 degrees of freedom. Therefore, a more compact representation is to multiply axis and angle together into a single three parameter vector [Pennec and Thirion 1997]:

$$\theta = \|\boldsymbol{\omega}\| \quad \text{and} \quad \boldsymbol{a} = \frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|}$$

This is known as the matrix exponential, one of the many different formulations of the exponential map, introduced to computer graphics by Grassia [1998], which maps $\Re^3$ into $SO(3)$ by summing an infinite series of exponentiated skew-symmetric matrices. To calculate the matrix rotation R from $\boldsymbol{\omega}$ the infinite series is generally evaluated using the *Rodrigues' Formula* [Murray et al. 1994] as follows:

$$R = e^{[\omega]_x} = I_3 + \frac{\boldsymbol{\sin}\theta}{\theta}[\boldsymbol{\omega}]_x + \frac{1-\boldsymbol{\cos}\theta}{\theta^2}[\boldsymbol{\omega}]_x^2$$

where I_3 is the 3x3 identity matrix (i.e. null rotation) and $[\boldsymbol{\omega}]_x$ is the following anti-symmetric matrix:

$$[\boldsymbol{\omega}]_x = \left[\begin{pmatrix}\omega_1\\ \omega_2\\ \omega_3\end{pmatrix}\right]_x = \begin{pmatrix}0 & -\omega_3 & \omega_2\\ \omega_3 & 0 & -\omega_1\\ -\omega_2 & \omega_1 & 0\end{pmatrix}$$

This minimal representation, favoured in recent motion capture work [Bregler and Malik 1998; Molina-Tanco and Hilton 2000; Alexa 2002; Lee and Shin 2002], is not one-to-one, since the same rotation can be represented by infinitely many vectors, whose magnitudes differ by 2π. If we assume that $|\theta| \le \pi$ then the exponential map becomes one-to-one, and it can be shown that the true distance between two rotations R_a and R_b is approximated by the Euclidean distance of their axis-angle vectors [Pennec and Thirion 1997; Molina-Tanco 2003]:

$$d(R_a, R_b) \approx \|\boldsymbol{\omega}_a - \boldsymbol{\omega}_b\| \quad \text{for} \quad \|\boldsymbol{\omega}_a\| + \|\boldsymbol{\omega}_b\| << 2\pi$$

Therefore the distance between two character poses P_1 and P_2 becomes:

$$d(P_1, P_2) = \sum_{j=1}^{J} w_j \left\| \boldsymbol{\omega}_1^j - \boldsymbol{\omega}_2^j \right\|^2$$

$\boldsymbol{\omega}_1^j \in \Re^3$ is the compact angle-axis vector at the j-th joint of pose P_1
$\boldsymbol{\omega}_2^j \in \Re^3$ is the compact angle-axis vector at the j-th joint of pose P_2

This gives a computationally efficient distance metric that is invariant with respect to rotations and translations (supporting linear dimensionality reduction techniques such as PCA). Unlike Euler angles, the distance between rotations is directly computable, the mean of a set of rotations can be computed, and a lower-bound to the true distance can be calculated [Molina-Tanco and Hilton 2000]. Interpolation can use fast Euclidean interpolants although optimality is only guaranteed if successive rotations are not too different from each other. Since motion capture keyframes are densely spaced in time, this is usually not a problem. These inherent qualities in the angle-axis parameterisation make it an effective candidate for our matching purposes.

Quaternions

Quaternions can also represent rotations [Shoemake 1985; Shoemake 1991; Barr et al. 1992]. A quaternion q is a complex number with three imaginary components:

$$q = w + xi + yj + zk \qquad \text{with} \quad w, x, y, z \in \Re$$
$$\text{and} \quad i^2 = j^2 = k^2 = 1$$

Often q is written as a 4-tuple $q = (w, x, y, z)$ or $q = (w, \boldsymbol{v})$ where w is the real part and $\boldsymbol{v}$ is a 3-vector containing the imaginary parts. The set of all unit-length quaternions where $\|q\| = 1$ forms the set of 3D rotations $SO(3)$ and there are no singularities in the mapping of unit quaternions to rotations. The quaternion formulation of a rotation about an axis $\boldsymbol{a} \in \Re^3$, with $|\boldsymbol{a}| = 1$, and by angle θ can be represented as [Shoemake 1985; Kim et al. 1995]:

$$q = \left(\boldsymbol{cos}\left(\frac{\theta}{2}\right), \boldsymbol{sin}\left(\frac{\theta}{2}\right)\boldsymbol{a} \right)$$

The angle-axis vector representation and unit quaternions are related by the exponential and logarithmic mapping. Given $\boldsymbol{\omega}$ is the angle-axis, then if $q = (0, \boldsymbol{\omega})$, quaternion exponentiation gives a quaternion of the unit length in the closed-form expression [Lee and Shin 2002]:

$$e^q = \boldsymbol{exp}(0, \boldsymbol{\omega}) = \left(\boldsymbol{cos}\|\boldsymbol{\omega}\|, \frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|} \boldsymbol{sin}\|\boldsymbol{\omega}\| \right)$$

Note the mapping is not one-to-one, so we limit the domain to $\|\boldsymbol{\omega}\| < \pi$.

Since antipodal equivalent points $q = -q$ represent a two-to-one mapping to the same rotation in $SO(3)$, the angular distance between two quaternions q_1 and q_2 is defined as:

$$dist(q_1, q_2) = \boldsymbol{min}\left(\left\| \boldsymbol{log}\left(q_1^{-1} q_2\right) \right\|, \left\| \boldsymbol{log}\left(q_1^{-1} (-q_2)\right) \right\| \right)$$

where $\log\left(q_1^{-1} q_2\right)$ is the constant angular velocity and the inverse of q is $q^{-1} = (w, -x, -y, -z) / \left(w^2 + x^2 + y^2 + z^2\right)$. $dist(q_1, q_2)$ is called the geodesic norm which is the shortest path between two points on the sphere and (contrarily to Euler angles and axis-angles) corresponds precisely to the shortest path between the two corresponding rotations in rotation space.

The distance between two quaternion-based character poses P_1 and P_2 then becomes:

$$d(P_1, P_2) = \sum_{j=1}^{J} w_j \min\left(\left\| \log\left(\left(q_1^j\right)^{-1} q_2^j \right) \right\|, \left\| \log\left(\left(q_1^j\right)^{-1} (-q_2^j) \right) \right\| \right)$$

$q_1^j \in \Re^4$ is the quaternion at the j-th joint of pose P_1
$q_2^j \in \Re^4$ is the quaternion at the j-th joint of pose P_2

Linearly interpolating on the quaternion sphere is optimal using *slerp* (or spherical linear interpolation) introduced by Shoemake [1985]:

$$slerp(q_1, q_2, u) = \left(\frac{\boldsymbol{sin}\left((1-u)\theta\right)}{\boldsymbol{sin}\,\theta} \right) q_1 + \left(\frac{\boldsymbol{sin}(u\theta)}{\boldsymbol{sin}\,\theta} \right) q_2$$

where $\cos\theta = q_1 \cdot q_2$ and u changes uniformly from zero to one. Spherical cubic interpolation (also known as *squad*) [Shoemake] and other more sophisticated schemes [Rammoni; Buss and Fillmore 2001] are other alternatives providing smoother interpolation (Figure 83).

Figure 83: **Spherical interpolation**. 3D Visualisation [Dam et al. 1998] of the interpolation curves on the 4D unit sphere over six rotation keyframes using linear *slerp* operator (Left), its smoother cubic equivalent *squad* (Middle) and the poorer direct linear interpolation of Euler angles (Right). Notice the Euler curve goes intersects the 3D unit sphere meaning that the interpolated points are not unit quaternions.

These characteristics make quaternions an appealing choice as our base matching primitive. However, these advantages are offset by the inflated computational costs. Omitting the higher dimensionality due to a 4D representation, the sheer number of necessary geodesic calculations to determine character pose distances becomes a bottleneck during matching. If scale-invariant matching is selected, we cannot use Euclidean interpolants and thus even the linear *slerp* interpolation slows the candidate resampling process by a factor of 50 to 100.

APPENDIX C

MATCHING ALGORITHMS

```
function FastLB_SubsequenceNN(query Q,
                              temporal threshold δ,
                              joint weights w,
                              database MBR Index I)
    BestTimeSeries = null;
    BestMatchVal = Infinity;
    BestOffset = null;
    minUpper = Infinity;
    MBE_Q = constructMBE_δ(Q);
    MBR_Q = createMBRs(MBE_Q);
    Priority Queue queue ← ∅; //keeps one entry per candidate sequence sorted
                              //according to the distance estimate

    for i = 1 to |C|, the number of sequences in database C
        MBR_Ci ← I[i]; //retrieve candidate MBRs from memory
        for j = 1 to |MBR_Ci|-|Q|, every valid offset in MBR_Ci
            queue.push( MINDIST_w(MBR_Ci[j:j+|Q|-1], MBR_Q) , i, j);
    queue.sort;

    while queue not empty
        (minDist , i, j) ← queue.pop;
        if minDist > minUpper
            continue;
        if minDist > BestMatchVal
            break;
        C_i ← C[i]; //retrieve original sequence from disk
        dist = wDTW_δ,w(Q, C_i[j:j+|Q|-1]); //exact distance
        if dist < BestMatchVal //eliminate false alarms
```

```
                    BestTimeSeries = i;
                    BestMatchVal = dist;
                    BestOffset = j;
              //Update minUpper
              maxDist = MAXDIST_w(MBR_Ci[j:j+|Q|-1], MBR_Q);
              if maxDist < minUpper
                    minUpper = maxDist;

        return(BestTimeSeries, BestMatchVal, BestOffset)
```

Table 8: Pseudo-code to optimally find best subsequence match using lower and upper-bound estimates.

```
function FastUB_SubsequenceNN(query Q,
                              temporal threshold δ,
                              spatial threshold ε,
                              database MBR Index I)
      BestTimeSeries = null;
      BestMatchVal = -Infinity;
      BestOffset = null;
      MBE_Q = constructMBE_δ,ε(Q);
      MBR_Q = createMBRs(MBE_Q);
      Priority Queue queue ← ø; //keeps one entry per candidate sequence sorted
                                //according to the similarity estimate

      for i = 1 to |C|, the number of sequences in database C
            MBR_Ci ← I[i]; //retrieve candidate MBRs from memory
            for j = 1 to |MBR_Ci|-|Q|, every valid offset in MBR_Ci
                  queue.push( E^lccs(MBR_Ci[j:j+|Q|-1], MBR_Q) , i, j);
      queue.sort;

      while queue not empty
            (SimEstimate , i, j) ← queue.pop;
            if SimEstimate < BestMatchVal
                  break;
            C_i ← C[i]; //retrieve original sequence from disk
            Sim = LCSS_δ,w(Q, C_i[j:j+|Q|-1]); //exact similarity
```

```
            if Sim > BestMatchVal //eliminate false alarms
                  BestTimeSeries = i;
                  BestMatchVal = Sim;
                  BestOffset = j;

      return(BestTimeSeries, BestMatchVal, BestOffset)
```

Table 9: Pseudo-code to optimally find best subsequence match using upper-bound estimates under *LCSS*.

```
function BruteForce_ScaledSubsequenceNN(Q, maximum scaling factor sf_max)

      OverallBestTimeSeries = null;
      OverallBestMatchVal = Infinity;
      OverallBestScaling = null;
      OverallBestOffset = null;

      for i = 1 to number of time series in database C
            for every valid offset in C_i, j = 1 to |C_i|-|Q|
                  [dist, scale] = TestAllScalings(Q,C_i[j:j+|Q|xsf_max -1])
                  if dist < OverallBestMatchVal
                        OverallBestTimeSeries = i;
                        OverallBestMatchVal = dist;
                        OverallBestScaling = scale;
                        OverallBestOffset = j;

      return(OverallBestTimeSeries,OverallBestMatchVal,
             OverallBestScaling, OverallBestOffset)
```

Table 10: Pseudo-code to find best scaled subsequence match over a database.