Compression and Indexing of 3d Motion Data for Animation

by

William John O'Kane, BSc

Dissertation

Presented to the

University of Dublin, Trinity College

in partial fulfillment of the requirements

for the Degree of

Master of Computer Science in Interactive Entertainment Technology

University of Dublin, Trinity College

October 2008

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

William John O'Kane

September 11, 2008

I would like to dedicate this work to my Granny,

Madge Bradley.

Acknowledgments

Foremost I would like to thank Dr Rozenn Dahyot for her advice, good humour, help and above all patience. I thank Dr Steven Collins for some useful advice, Dr Neil Lawrence for some useful code and finally I would like to thank my parents for supporting me while doing this work.

William John O'Kane

University of Dublin, Trinity College
October 2008

Compression and Indexing of 3d Motion Data for Animation

Publication No. ____________________

William John O'Kane, MSc IET
University of Dublin, Trinity College, 2008

Supervisor: Rozenn Dahyot

Motion capture databases are increasingly a topic of research with much work going on in the areas of motion retrieval, compression, synthesis and motion processing. We investigate some questions with regards to the application of compression techniques and an indexing technique. The data transformation techniques in question are those of discrete Fourier and wavelet transforms as well as the application of principal components analysis (PCA) and probabilistic principal component analysis (PPCA) to study the underlying low dimensional motion data.

Our questions revolve around the ability of PCA to aid in the compression of classes of motion and why or how it works better under certain circumstances. We offer the use of a positional representation, aligned in global orientation, to aid compactness of representation and suggest this performs better than hierarchical channel based representations of the motion data. We also compare the effectiveness of either Fourier or wavelet transforms for compression on different classes of motion and find the Discrete Cosine Transform (DCT) to perform well under most circumstances. We create a compression routine based on a combination of PCA and DCT and study it with regards to compression of different classes of motion.

Finally we consider classification using PPCA instead of manual techniques and note how well it works for all types of format but that it performs best using our normalised and aligned positional format. We suggest future possible uses of this classification technique in compression.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Motion capture is commonly used to record human motion in various entertainment fields. The motions, motion clips, are often stored for reuse for future projects creating large databases containing human motion. This prompted new research that studies motion capture. Synthesis, previously used to fix up animations or blend known ones, is now being augmented with data from databases [1] to help create entirely new motions and new models of human locomotion are being extracted from the databases [2].

Browsing these databases manually requires slow inspection of each motion, possibly sped up by meta-data [3] or by pseudo representations [4] of the motion. Ideally adding a new motion to the database would not require extra tagging or categorisation of the motion to aid the queries. For this reason there is active work on attempting to "Query by example", a common paradigm in audio, images and text searching. In query by example the input is a motion clip for which we wish to find a qualitatively closest match to return as a result to the query. This has uses for many applications involving synthesis and for simple human use. An interesting application is that of extracting low resolution information from video and recasting it as a query problem for a motion capture database [5]. This could be used as a means to interactively select actions or replicate the motion on a virtual avatar.

In the previous virtual avatar example an obvious direction is for games consoles and the memory allowance on such consumer hardware is very limited. For this and

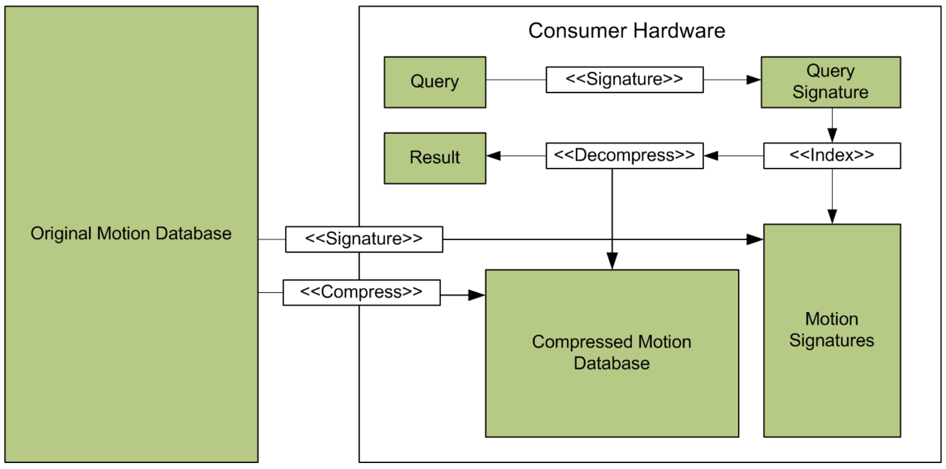


Figure 1.1: Example architecture. Signature, Index, Compress and Decompress represent functions while the shaded areas represent data.

for other reasons compression is important on large databases. It's also important because it allows us to try define the true nature of the underlying data which is useful for further work on synthesis or querying. As a high level goal we imagined a simple system that defined how a compressed database suitable for querying might work on a memory constrained architecture. As you can see in fig 1.1 both the compressed motion database and motion signatures (indexing structures) are transformations of the original database which is ultimately discarded. We are interested in the potential transformations one might make to the data and what format it takes. We had several transformation techniques and query techniques in mind starting out. To explore the best transformations and to consider how best to perform querying we defined several questions that we would like to try answer. My personal knowledge of the given techniques was non-existent and my knowledge of the research area lacking. So a large motivation in the work of this dissertation was to grow that knowledge and apply some of the research techniques for better learning. The next section covers the questions.

1.1 Questions

PCA (see section 3.2.2) in theory should aid compression due to its ability to reduce dimensionality of data, but it doesn't come for free. At what point can we suggest that using PCA is a good idea. Are there different data formats for which PCA is less useful or classes of motion for which PCA won't work as well? To test this we perform compression on various sizes of motion in the multiple formats, with and without using PCA. From this we hope to gain an understanding of the benefits or not of using PCA.

Of the two techniques of interest for compression of time series data (we might call them temporal compression techniques), discrete wavelet (see section 3.2.2) and Fourier transforms (see section 3.2.2), are there preferential choices for certain classes of motion or in general? To test this we perform a PCA and temporal based compression using various human chosen classes of motion files with either temporal transform. From this we hope to identify if there any cases where one technique is clearly better than the other.

PPCA (see section 3.2.4) is a technique that allows us to classify the likelihood that a pose belongs to a given trained set of poses. How good are the results of using this technique? To test this we train several classes of motion and then use test motions in an attempt to observe the classification in action.

How good in terms of bytes per frame or ratios of compression do the better techniques perform overall? When performing the above tests we hope to find some answers and insights by asking this question.

1.2 Guide

In state of the art, 2, we cover the most recognised papers for the separate topics of querying, synthesis, compression and processing of motion capture databases. We provide some commentary and try to show some patterns in approach to the solutions. In design and implementation, 3, we quickly cover what tests we devise to answer the questions, the permutations of formats and techniques and how to measure error. Then we get into the gritty details of the data formats and their transformations, compression, recognition including some of the maths involved. This takes up the main part of the dissertation, as it did my time. Evaluation, 4, is a section providing the results of the tests with attempted reasonings for the numbers. Finally conclusions and future Work, 5, reviews the work done, the results and suggests what is needed to advance.

Chapter 2

State of the art

In this chapter we discuss the current literature on motion capture research. One might split up the research in many ways and we choose to categorise it by four overlapping topics, "querying for motion" or motion retrieval, "motion synthesis", "motion compression" and "motion processing".

2.1 Overview

Querying for motion is a desirable tool for human users of a database as the databases become larger and larger. To observe all the animations a user would potentially need to watch hours of motion capture. Querying may be done by key words or labels if the motion capture has been tagged with the relevant information. Unfortunately interpretation is often flawed, quality of labeling inaccurate or missing and desired motions embedded within other motions. For a human observer it is probably best if they can find better ways for browsing the database, through improved visualisations or querying by example motions. For an algorithm, the ability of being able to query by an example motion also clearly has uses for reuse in synthesis and thus compression and for classification.

Faster and more accurate queries are important and lead to one reason for motion classification. If we can model a class of motion and represent it using a small amount of data that differentiates it from other motions then we can do quicker searches. In developing classification we also improve our understanding of the types of motion data and this leads leads to some applications in synthesis and compression.

The range of motion for humans, while large is still finite and often repetitive. As the databases of motion information grow, the gap between the coverage of all human motion possibilities and those represented gets smaller. It is no surprise that data mining techniques are being used to try extract models of locomotion, to augment missing information, to sequence clips from the databases to generate larger motion clips or to blend clips together. Synthesis allows us to reuse the information already there in new ways.

For off-line work the databases being used may be very large and for real-time applications the amount of main memory is often limited. Typically motion capture contains many degrees of freedom at a high number of frames per second, 62 variables per frame at 120 frames per second is common for a human character with no fingers, toes or facial features. This overly represents the underlying motion and for compression we are interested in models of motion that can accurately represent characters using less information with just the noise left over.

Sometimes we simply wish to process the information in the database. Common problems are segmentation of data, annotation of data or the removal of noise. Segmentation is a useful tool for compression, classification and synthesis because it splits longer motions of connected data into smaller clips. Annotations, done on-line with human aid, can improve classification or human guided synthesis. Finally removal of noise can remove sensor data inaccuracies and aid synthesis and compression as well as giving us a paper case study into wavelets.

2.2 Querying for motion

Querying by example is by far the most common approach for querying. The key issue that arises is the disconnect between the query numerically and the query logically or temporarily. That is to say, if we wish to search for punches then all punches are considered valid logically, but numerically (via a norm based distance metric) a left handed punch may not be numerically close to a right handed punch. Temporarily we may also have a disconnect in that slow runs and fast runs will not match up although the human observer or even an algorithm for locomotion may want all runs in response. That is not to say that a norm based distance metric has no place, it does, but it must be considered with mismatches and false positives in mind.

In [6] we are given a discussion and solution to the temporal problem. A common solution to the problem of time differences between motions is to use Dynamic Time Warping (DTW), first introduced to motion capture by [7, 8] and commonly used in audio or in time series matching for databases. This slows down or speeds up a motion against a reference motion per frame such that time is still monotonically increasing, continuous and not distorted too much and such that the local minimum of the distance between poses is optimised. The Euclidean distance is then measured. This technique is improved upon by [6] by allowing a global scaling of the motion first and not forcing the two motions to have equal length. In a brute force approach all possibilities of scale within a min/max scale allowance would be tested by measuring the squared Euclidean distance between all frames of the smaller motion and the matching frames on the larger motion. For a search query this would also be repeated for all possible pairs of motion. To speed this up an exact lower bounding, “Bounding Envelopes” [9], is used to prune off all motions for which a match is impossible and optimisations are applied to speed up searching on the hard drive. DTW can then be performed as a post step.

[10] attacks the logical problem mainly by taking the results of a query (found using a norm based and time aligned, DTW, distance metric) and performing the queries again so that further nearby queries may be found. This continues until no further matches are found. To accelerate this they perform an extensive pre-computation stage to generate a “Match Web”. The match web is the result of performing a distance calculation of every frame of the database against every frame of the database with the results stored in a large 2d grid. The DTW algorithm is then used to find a minimal cost path from every cell in a bottom left to top right fashion in the grid such that the path is of at least minimum length and within an average distortion value. An extended valid region around the path defined to aid robust handling of aliasing and

minimum plateaus and bridging between paths is allowed should they be sufficiently close. The resulting graph can be stored using just the affected cells. Searching is an intersection of the motion indices on one side of the graph and paths that span both limits. A match graph represents the results of the multiple queries, their costs and time alignments. The multi-step querying approach is an interesting way of solving the logical problem, but the "Match Web" would require applications that can assume a lot of preprocessing of the data is allowed.

Both of the previous techniques focus a lot on time manipulation for better matching. But there is another approach which is very important and that is searching for correlations between configurations of poses. If we imagine a pose as a vector in a high dimensional space defined by the values of joint positions or rotations then an area where poses cluster suggests similarity. Time correlations are still valid of course, but time issues become less important. For example [11] provides an approach where motion sequences are classified pose by pose by the cluster they are associated with and call this a "cluster transition signature". To do this principal markers are determined via Principal Feature Analysis (PFA) [12] and used in a hierarchy of clusters that represent the motion to within an error tolerance and in a maximum of d dimensions. The centroid of each cluster is used in a decision tree classifier to guide an input pose to its appropriate cluster. The technique also extends to motion compression and estimation of a pose from an input pose of less variables.

[13] uses a Principal Component Analysis (PCA, features in PFA and regularly in motion capture papers) on pose data defined by a hierarchy of modified quaternion joints. Weights are applied to each joint to help indicate importance of that joint, as defined by a measure of body mass influenced by the joint, when compressing to a signature. Different types of motions require different types of weights. Weighting is a common requirement for dealing with poses defined with variables that represent a hierarchy of joints. This is one of the motivating reasons why we avoid using an Euler or Quaternion representation later in the dissertation. We would rather avoid the weighting schemes and the non-linear nature of these representations that require non-linear solutions. The query motion and the database are projected onto the weighted PCA space. To speed up the indexing a "characteristic" pose is found in the query

motion. This is a prominent pose in the set of query poses. An Approximate Nearest Neighbour (ANN) search is used to find similar poses to the characteristic pose. The time indices of the poses are clustered, to reduce the number of poses and DTW is used on the means of the clusters to attempt motion matches. The "characteristic" pose is an idea that leads to problems in that it's hard to define a motion by just one pose and the clustering of time indices assumes that the results have fairly similar time alignments to each other and the query motion. This may not always be the case.

All the previous approaches stay close to the original representation of a time series of real values representing position and rotations but some novel ideas have arisen that consider an alternative representation. [14] focuses on efficiency by segmenting the motion capture database and converting the motion clips into a representation based on geometric features [15] and then clustering them. Then a curve peak matching algorithm is used to compare motions from a cluster with a query motion without using DTW. Geometric features are a set of features, as large as is deemed necessary, describing states like whether the head is above the shoulders or whether the left foot in front of the right and so on, for each frame of motion. One of the nice results of geometric features is that they help avoid numerical norm issues, the results are more likely to be logically sensible, and they are less connected to time. A sequence of motion segments becomes a sequence of indices, each pointing to one possible combination of feature values.

In a similar approach [16] suggests using a set of boolean geometric relational features, "relational motion features" for each frame discarding a lot of real numerical data in the process but retaining key identifying features of the motion. For example only 39 features are used to describe a full body pose. The motion is represented as a matrix of features. "Motion Templates" are trained from example motions by time warping and averaging the relational motion feature matrices from the different training motions (further iteration is performed on the same data to converge the results that are biased by the initial choice of reference motion template). Motions can then be classified by the motion template they match.

2.3 Motion synthesis

For synthesis from a database there are several active research main areas. One, best described by the term "motion graphs" and introduced by [17, 18, 1] splices small clips of data from throughout the database into a new sequence. This makes good reuse of the high quality motion capture data without actually knowing how to generate a specific pose and without preplanning the sequence of necessary motion clips. The motion capture data is preprocessed to find poses that are similar enough, within a threshold, to allow for a natural looking transition between the motions. The process used for finding transitions by [1] is like that of the match web in [10], i.e. comparing distances between a pose and all other poses in a grid and noting the minimums. A graph is built from the motions and the transitions and then the problem of creating new motions is one of searching the graph for an optimal solution to the synthesis problem posed (including constraints). Choice of constraints in the query, how graph weights are chosen and constraints solved defines the quality of the output. The shortest path is not always the right choice.

Motion blending is the idea of taking two motion clips and combining the numerical data for poses in one or more motions using a weighted blend. This is a ubiquitous idea in motion editing packages and in realtime applications like computer games. This author previously worked professionally on such work. The restrictions on motions that will blend properly are rather high. They must be aligned in global position and rotation (or have it removed completely and synthesized later), they must have similar cadence and length, bar global scaling tricks, and they must still be from compatible motions, such as a run motion and a jog motion with the footstep order matching. [19] suggest some improvements to typical motion blending to try increase the range of motions that can be blended using a set of techniques labeled "registration curves". In this root position and rotation on the x-z plane is extracted from each motion to create an alignment curve, DTW is performed upon the motions and overlapping foot constraints (annotated constraints are required, even if via an automatic approach) are averaged together. The final registration curve will align the root orientation and blend the pose by motion weight as well as indicating where and when the foot constraint solver for foot placement should occur.

More recently work has been done on augmenting a small set of inputs (markers) representing under described motion capture data with the data in a motion capture database. This has great uses for action selection in interactive applications allowing a casual amount of preparation or for motion capture data synthesis using only cheap motion capture setup. [20] describes a system where they track some inputs and the orientation of a user at real-time. A local linear model (PCA) is made from similar data found in the motion capture database, the mahalanobis distance helps indicate which of the potential poses is probable for the local model. Previously synthesized motions are used to indicate how smooth the potential pose would be. Finally the potential motion capture pose is checked to see which pose is closer to the inputs. These terms are combined to help choose the next pose to use. A motion graph or "neighbour graph" is used to accelerate the search for potential matching poses in the database.

In section 2.2 we saw how time related techniques like DTW were swapped for time agnostic techniques like clustering poses. In synthesis we have a similar comparison for motion graphs, which are decided using temporal considerations and clustering in [5, 11]. Here the database is preprocessed such that the motion capture database becomes a hierarchy of clusters of motion segments that can also be represented as local linear models. Motion segmentation techniques [21] are used to split the motion database into smaller motions representing inherently low dimensional data. The motion segment is converted to a high dimensional "feature vector" which is reduced by PCA and divisively clustered using K-means such that cluster mean is within a tolerance (or the tree is further split in two if not). A local linear model is created for the leaf using the motion segments located there. A mapping to estimate the non-principal markers from the principal markers (found using PFA) is generated for the local model from the cluster poses using a least squares approach. When generating a pose from a few input markers it is necessary to find the proper local model to use. This is done by taking the principal markers of each pose in the local model and using them to train a classifier using "Random Forest" [22].

2.4 Motion compression

Advances in the understanding of the underlying motion data structure and dimensionality have prompted new techniques that exploit this to reduce the amount of data to represent motion capture with low error of reconstruction. One example that exploits both joint coherence via local linear spaces and temporal coherence via splines is [23] following on from other work previously mentioned. The assumption that generally holds true is that motion poses close to each other in time also often inhabit the same low dimensional space when transformed under (PCA, see section 3.2.2) and the key to good compression is to split the motion segments at the point in time before the number of dimensions required to represent a sequence of frames or poses starts to increase [21]. This means that only a few principal component vectors are required, i.e. the number of dimensions required to reconstruct the motion stays low. The low dimensional projected pose vectors are compressed on the time axis as splines. An important step is the normalisation of the pose data, what we later call alignment (see section 3.2.1), this removes the global rotation and translation from each pose vector to create a more compact set of principal component vectors. The alignment data is compressed separately using a spline based approach and feet markers are compressed via PCA but using a lower error tolerance. Finally for transitions between motion segments a weighted mixture of PCA reconstructions or overlap is used to help smooth the results such that glitches between different projections on the local spaces don't show up.

Special consideration of the feet is a common theme in compression papers, this is due to human sensitivity to contacts with the environment that break physical laws, such as penetration of the ground or putting a hand through a door knob. In particular the contacts that feet have with the environment are short sharp and significant in terms of signal change. In [24] we find that feet or other joints in contact with the environment are compressed using a special Discrete Cosine Transform (DCT, see section 3.2.2). This works particularly well because of the unique way the rest of the motion capture is compressed such that the values for foot joints are stationary for the motion segment upon touching the ground. First the motions are split into segments of a fixed length, k, then they are aligned by the global orientation and translation of the first frame, i.e. such that at the start of the segment the character is always at the origin and possibly moving away from it over the next k frames. The motion capture data is converted to a positional representation called "virtual markers". This stores the position at the base of a joint and two extra offset positions that effectively represent two axis vectors from the rotation matrix for the joint. These extra positions are stored relative to the joint position. Bezier curves of four control points, are fit to the trajectories of each virtual marker over the k frames, making 12 values per marker. The number of bones times 3 times 12 equals the size of a vector to represent the motion segment. Clustering is performed upon the vectors of motion segments and a PCA space for those vectors is created within a reconstruction error, typically only requiring a few principal vectors. Numbers of clusters, error and k are all input parameters. Motions are reconstructed using the projected coefficients, the cluster/PCA index, the few cluster principal vectors, cluster mean and Bezier curve decompression as well as the separate decompression of the environmental contacts such as the feet and the global orientation for the segment. Finally an Inverse Kinematics (IK [25]) fixup is performed upon the feet. Similar problems of discontinuity between PCA spaces exist as found in [23] and need to be solved.

A different approach, concerned with efficiency and the problem of needing to decompress a full pose described by the previous papers including all that may entail (typically at least a set of principal component vectors and cluster mean) is described by [26]. They highlight the case where one may need to access only a few joints of the motion, not the entire motion, which is typical for games applications which don't want to incur the cache hit of loading unnecessary data. So they focus solely on temporal compression and work with joint angle data. They use a cubic interpolating bi-orthogonal wavelet basis for their wavelet compression of each joint value over time. Error is calculated based on the difference between the original motion and compressed motion converted to joint positions and scaled by bone importance as decided by length of the bone. One way of compressing after a wavelet transform is to remove the smallest coefficients per time varying signal. An advancement is to remove the smallest coefficients across all the signals in a motion after weighting the signals such that joint angles and positions are given proper equal importance. The real optimisation comes from realising that bone importance changes depending on the motion. A heuristic based simulated annealing approach is used to choose the optimal set of signal weights for a given motion to remove the smallest coefficients with. Foot contacts are again given special consideration and fixed with an I.K. process.

Common to all three previous papers discussed is the special consideration given to how important a joint is perceived to be. Recognising that perceptual differences between the joints exist is important for maximising compression ratios. Recent work by [27] highlights areas of special consideration for perceptual based compression on motion capture data in particular contexts such as in crowds.

2.5 Motion processing

Motion processing is often used to transform the motion database to a more useful or desirable form for further work in synthesis, recognition or compression. One example that uses user defined annotation to try classify other motions for annotation and assist synthesis is [3]. The overall goal of the paper is to allow the user to paint a time line with annotations to indicate the motion sequence to synthesis, including lower level pose constraints and global position constraints. The transformative aspect of the paper is in preparing the motion capture database with annotations using a vocabulary of their choosing to suit the database, e.g. "walk", "jump", "turn". Annotations may overlap, it is up to the user to paint the training motion so. Using a small set of example annotated motions a Support Vector Machine (SVM) classifier was built to provide for fast classification of the rest of the database.

A paper that has been of importance to papers in previous sections is [21]. This paper deals only with automatically segmenting motion capture databases. Three approaches are studied for splitting motions by high level behaviours for which one might assign a verb. The motion data is the entire database represented as one sequence of frames. Each pose is a typical hierarchy of positions and quaternion rotations with global position and orientation removed. The first approach is a PCA based approach with each pose as a vector. The assumption is that motions of similar behaviour take up a space of low dimensionality as defined by the number of principal components (dimensions) needed to represent the motion within an error tolerance. For each segment, a fixed number of frames are first used to create the local PCA space and then the number of dimensions to represent it within an error are calculated. New frames in the motion sequence are added incrementally and PCA is performed again using the same number of dimensions. If the error rises slowly there is no new motion, but if it changes quickly then a motion segment has been detected and the process is restarted for the upcoming frames of motion. The second approach uses Probabilistic PCA (PPCA, see section 3.2.4) to calculate a value for the probability that a given pose is within a gaussian distribution of a set of principal components. In increasing stages of frames a local PCA space is created and the distances for the frames, as defined by the Mahalanobis distance that characterises the probability of the space, are produced. Peaking finding in the distances over the frames is used to segment the motion. The third approach uses a Gaussian Mixture Model (GMM) to break up the motions when consecutive sets of frames belong to different Guassian distributions. The assumption is that behaviours cluster via their poses. Results showed that the PPCA approach worked best due to it's ability to track joint variance and dimensionality.

A final example of motion database processing is that of smoothing motions to remove noise. In [28] they apply a wavelet smoothing approach to quaternion motions. This performs in similar way to wavelet compression in that it modifies the smallest wavelet coefficients after a multi-level wavelet decomposition (see sections 3.2.2, 3.2.3), they chose Daubechies_10 as their wavelet due to its smoothing properties. The thresholding can be hard or soft, meaning it may set the values that don't make the threshold to 0 and it may reduce values by the threshold value if they do. The step they take for working with quaternions is to calculate the (discrete) angular velocity defined by the motion and perform the smoothing on this before integrating the angular velocity back into a quaternion. This guarantees the result is a unit quaternion and because only noise is being removed the rotation trajectory should still be valid.

2.6 Summary

Of most interest to this dissertation is the various ways that these papers have attempted to tackle the issue of data representation and similarity matching. Two main approaches stand out for trying to deal with variable length and numerically different motions which we would consider logically similar. The first is to warp and scale the data on the time axis using local and global optimisation approaches such that the data lines up and then perform an norm based comparison. For any application concerned with speed and memory size this approach seems slow and requires all of the motion files to work with even if extensive preprocessing can help. The second is to transform the data into a new representation more suitable for the application. For just comparisons, e.g. querying and indexing, we see the benefits of geometric features and for compression or retrieval we see benefits of local linear models often estimated by some sort of clustering approach. The key observation is that clusters represent locality of motion clips in the same way that time warping can do, without the need to warp. Our preference is to transform the data, even if only projecting onto a local space via PCA. This also allows for further potential to compress, remove dimensions or warp on the projected time series data.

Synthesis has experienced an increased amount of activity now that the idea of augmenting synthesis with motion capture databases is popular. Motion graphs suggest splicing together motion clips to meet problem constraints of moving an avatar realistically. Practically speaking, no game would consider such an approach, defining an avatar path or set of actions n seconds into the future is a complicated time consuming A.I. problem and there's every reason to preplan and define the allowed set of motions an avatar has in a design stage, or at least off line. Blending on the other hand is very practical and the "registration curves" approach is a useful collection of techniques for avatar locomotion blending, provided the cost of performing DTW is not prohibitive - this would make sense in an animation tool application. Of the two techniques for quickly synthesizing a motion capture pose for an input pose of only a few markers we would again prefer more the cluster based approach as a means of aligning similar motion in the preprocessing stage. This type of synthesis is perhaps the best case study of state of the art since it must find similarities in the database using a limited set of input signals and reconstruct from the database a full pose, also implying that it cannot throw data away like typical indexing techniques may do.

Compression is usually considered a simpler problem to reason with if we only consider numerical value but it is surprising how much perception affects the work here. In all cases the gains were achieved by weighting in some way the importance of particular perceptually important joints. The first approach followed the synthesis approach of finding locally linear models after segmentation and then doing dimensionality reduction and finally temporal compression on the remaining time varying projected values, with special consideration for feet and overall positioning of the character. This is closest to our own work. The second approach is a little similar but changes the problem such that the ideal circumstances for compression on the feet is introduced, that is short signals where the values may become static (foot planted) and compress faithfully using DCT. Finally a speed conscious approach is provided in a wavelet compression scheme that uses dynamic programming to try find the optimal selection of coefficients for each motion at a time by altering the bone weights.

Understanding the data in a database is important and tools to unlocking the potential are important. Better annotations are invaluable but time consuming to generate so any approach that can automatically classify from previous annotations is very useful. Perhaps the most important work was that of good automatic motion segmentation, as much as for what it tells us and for how much it helps us fold the similar pieces of database together.

Chapter 3

Design and implementation

In this chapter we describe the design of the tests and the implementation of the various data formats and techniques.

3.1 Design of tests

We set out to answer some questions about the nature of motion caption data formats and techniques that could be applied to the compression and recognition of poses. So the first step in the implementation was to create a framework for loading motion capture files from their inherent "ASF/AMC" format and to consider what possible numerical representations might be used. At the same time several compression techniques and a recognition technique were investigated. These are discussed in detail in the implementation section 3.2. Eventually we created a permutation of data formats and the interplay with the various compression techniques. These are shown in fig 3.1. With these we would hope to gain a better understanding of the nature of motion capture data and which formats are better. For example it is suggested [24] that the non-linear and hierarchical nature of "channels" makes conversion to a positional representation a sensible choice, while [23] suggest that "normalisation" (confusingly the same thing we refer to as "removing root motion" and different from our normalisation routine) should aid compression via Principle Component Analysis.

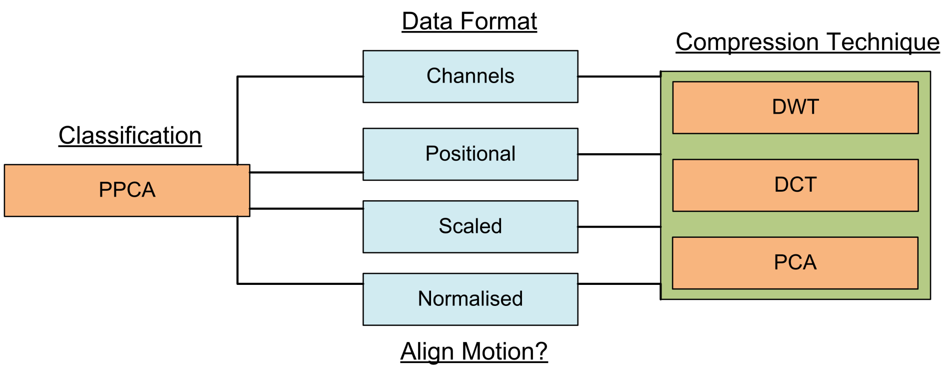


Figure 3.1: Combinations of data representation and techniques

3.1.1 Error and bytes

When performing lossy compression one reduces the number of bytes and the error goes up, or one increases the bytes and the error should go down. Either way both are linked and can be used to help gauge how well the compression is performing or give a guide as to how well presented the data is. When considering motion files we are dealing with an array of poses or frames sampled at a fixed rate, typically $120hz$. Because different motion files have differing lengths it is not meaningful to just use the total number of bytes when considering a motion. Instead we divide the total bytes by the number of frames of animation to create an average number of bytes per frame. Bytes per frame is often used with a fixed error metric. E.g. Fix some error parameter at a value of 0.5 and calculate how many bytes it takes to compress for this error.

Alternatively we may consider the error given a fixed amount of bytes. This can be useful for time critical applications where the bandwidth allowed for the data is fixed within some maximum. In particular if we know the amount of bytes being used for a “packet” of work one can optimise indices of coefficients, hardware caches and local memory buffers to accept the “packet” optimally. But error may fluctuate. Qualitatively it is easier to indicate a fixed error which must be met numerically as a means of describing quality of compression or how well represented a motion is than it is to indicate a fixed number of bytes. Typically we will use a L_2 Norm of some sort to describe the error. Again the length of the motion is variable per motion so we must develop an average metric per frame to normalise results.

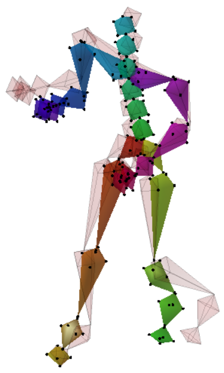


Figure 3.2: Test mesh and vertices (colour) with original mesh (faded).

Initially during this project error was defined as the average L_2 norm between the frames of the positional representation (see section 3.2.1) of a motion. Any of the data formats could have been chosen, but the positions were considered the most natural because one half of the data was made up of global positions of joints, the other half being global orientation vectors representing part of the bone orientations. This makes the data in the positions somewhat close to the data one would use to actually view the final motion. Channel data on the other hand contains Euler angles which are non-linearly associated with the resultant motion, as well as being hierarchical in nature, so the importance of an error varies per bone. Bones at the base of a "skeleton" having more effect on the outcome of the global pose than child bones and this itself varies depending on the orientation of the child. E.g. if a character has their arms outstretched then the twisting rotations at the base of the spine are more prone to causing error than a character with their arms tucked into their pockets, [26] describes this effect in detail and solves it using a simulated annealing approach.

Remembering that the positional representation is made up of global positions and global unit orientation vectors it's clear we don't suffer hierarchical issues. On the other hand the orientation vectors are still problematic since they don't change size depending on the character size, and even if they do we are implicitly suggesting the amount of error that should be weighted towards the orientation data. To avoid this we moved to a skin error approach. Instead of just converting to the positional representation to check error we go one step further and skin a mesh. The mesh is a collection of faux bone representations made up of six vertices per bone and shown in fig 3.2. The vector, a “skin”, we compare against each for frame is thus made up of 540 dimensions (30 bones × 6 3D vertices). The average error for the motion is shown in equation 3.1.

$$\epsilon(Motion) = \frac{1}{N-1}\sum_{i=1}^{N} ||skin_i - skinOrig_i|| \tag{3.1}$$

Using the skinning error we really can't get much closer to the visual, or perceptual, representation numerically other than to specifically skin it with the mesh that will be used for rendering. One of the main benefits is that we are now measuring the error of rotations better, a small twist of the wrist is given less importance than the outward swing of a leg, even though both values are represented by the sized unit vectors, or same radian Euler values in position or channel representations. Ideally we would like to have a meaningful way to transmit this knowledge back into the compression processes but this is a somewhat harder problem. Many of the issues are context and view dependent. When compressing it is not common to have prior knowledge of these constraints. When we do, such as whether the character is male or female or whether the motions are to be only used at low resolution, it would make sense to take into account the perceptual guide lines.

3.2 Implementation

In this section we cover the implementation and use of the data formats as well as the compression and classification techniques.

3.2.1 Data formats

The data for this project comes from the CMU Motion Capture Database [29]. The most common format provided is the ASF/AMC format. These represent a skeleton definition file, Acclaim Skeleton File, and motion data, Acclaim Motion Capture data, respectively. There are over one hundred sets of data, each set containing one ASF file and multiple AMC files, each set presumably coming from one motion capture session. The files are loaded in Matlab using the Motion Capture Toolbox provided by [30] for which we are grateful.

Motions are stored as an array of frames, like a frame of video. Within each frame, or pose, are the variables that go together to make up the frame. In AMC files the motions are interpreted relative to information stored in each relevant skeleton. The motions are multiple "channels" of data, each channel representing a degree of freedom for a joint (bone). The root joint or bone contains three degrees of freedom for three dimensional translation and three Euler angles representing rotation around x, y and z axes. Further bones may have up to three Euler angle degrees of freedom, but no translational degrees of freedom.

Apart from the root bone these degrees of freedom are relative to the degrees of freedom information defined by the skeleton. So the skeleton defines an original set of values per bone "the base pose" in a world or model space, and then the the motion file contains further offsets to the base pose to create a "frame pose", for each frame of the motion. The Euler angles for the base pose and offsets for the frame pose are not defined such that they can simply be added together, they must be converted to rotation matrices and then multiplied. The order in which to apply the Euler angles is defined by the skeleton. Using this order, each angle is converted to a matrix representing rotation around the given axis and then multiplied together.Because ACM files are defined using a hierarchy of bones the rotations of the joints must be concatenated to recreate the world space rotation.

To interpret these values we would like, at some stage, for each bone to be relative to each parent bone, a local coordinate system for that bone (bone space) so that the offsets per bone can be calculated. Since only the root bone has any translational degrees of freedom the remaining bones get their offsets defined as a constant "direction" and "length" described in the skeleton. These offsets are considered to be in the coordinate system of that bone and define what translation the next bone starts from.

Because each bone is relative to a parent it is necessary to consider the bone rotations and translations of each parent in the skeleton hierarchy if we want to know the final (global) translation of a particular bone. The formula is as follows:

$$\begin{aligned} R &= C^{-1} * M * C * R_{parent} \\ T &= T_{parent} + V * R \end{aligned}$$

R is the world rotation of the current bone, C is the base pose rotation, M is the offset rotation, T represents translation for the bone and V represents the current bone offset. The addition of C is a little confusing, why not simply define M as offset from the parent rotations instead of multiplying the rotational coordinate system defined by C, rotating by the offset M and undoing the C rotation. This was done so that motion, M, could be defined separately to the coordinate system defined by the parent motion and separate to the bone direction. In fact, ignoring C the bone offset is not necessarily aligned with any of the axes, R or C. T is always calculated as the end point of the current bone, an addition of the parent's end bone position and the current bone offset rotated by the current bone rotation R. A slightly different representation is to consider the bone translation as follows:

$$T = T_{parent} + V_{parent} * R_{parent}$$

In this case each joint is only calculating the offset for the start of the joint. The responsibility for calculating the position at the end of the current joint is thus deferred to the child joint. Presumably this approach to calculating the translation is not taken because end joints would need to have a special case for calculating the position of the end of the joint. Either way works.

If we are performing the calculations above we move from a set a hierarchy of joints defined by channels to a set of joints defined by a rotation matrix and translation vector in model or global space. It is very simple to draw lines between joint translations to visualise the skeleton. Just using the translations may be enough in certain cases, depending on the application. E.g. if we simply want to draw a stick-man, then the translations are adequate. For classification translations may also be enough. But if we only retain the translations there will not be enough information to revert back to a full pose description suitable for rendering deformed meshes, the rotation matrices will have been lost. So in the compression scenario we must also keep the rotations.

Matlab matrices

All the different data formats to be described are stored in Matlab as two dimensional matrices, R^{NxM}, where N is the number of samples and M is the number of variables, or dimensions, for the given representation. So each column represents a one dimensional time series of data and each row represents a pose.

Channels

The channels have been described before and these values are relative to the base pose space. As noted before channels make use of a matrix C, a pose space matrix defined by the skeleton within which the motion data is rotated. In the CMU database the C matrices for each skeleton are within a negligible error when compared with the C matrices of other skeletons. This fortunate circumstance should allow for better performance of comparisons between channel data. The number of dimensions used to represent each frame of motion is 62.

Positions

This is the channel data, after conversion to positions. The question of how to store rotations arises. Numerically Euler angles are very different to positional data, having non linear effects. Ideally for a positional representation we would like to use only positions. So a technique similar to that described by [24], "virtual markers" is used to store enough data as positions to reconstruct the rotation matrices represented by the rotational data. We examine the rotational matrix for a particular bone and create two vectors that represent the x-axis and y-axis for the matrix and store them as a part of the positional data. This differs slightly from the virtual markers approach in that we don't care to store them in a global or model space, and instead keep them as simple axis vectors. This leaves us requiring 279 dimensions to represent a frame of motion, that's 31 joints by two rotation vectors and a position.

We then take this a step further and align all rotation matrices such that their x-axis is aligned with the bones. The direct conversion from channels does not provide this. In fact if all final rotation matrices for a frame are set to the identity matrix we will recreate the the bones in their base pose. A faux set of matrices is created to mimic the differences between the identity matrices and the bone alignments in the base pose. We say faux because the original skeleton file only gives us a global vector to define the base pose orientation for a joint, so some tricks and assumptions are used to generate a matrix from this. These matrices are then applied to the results of the rotation matrices calculated from the channels to give us the x-axis aligned motion. With the rotations aligned with the x-axis we can now drop one of the two rotation vectors, since it can be recalculated later as the subtraction of the parent and child bone translations. This leaves us with 186 dimensions. A final optimisation comes from realising that the global root rotation has been transferred to the remaining joints as a part of the conversion process and is no longer needed. The same is true for the root translation, but it is still needed to help calculate the x-axis rotation vector, so we keep it. This gives us 183 dimensions per frame.

Aligned positions

Global translations and rotations of the root joint have a large effect on the final positions of all the remaining joints and virtual markers. The variance of the global movement of a character often dwarfs the variance of movement for all other position values representing the character relative to the root position. Further, the global position of the root joint is often unrelated to the underlying nature of the movement. For example a character walking in straight line is still walking regardless of where in the world they are positioned. Arguably the global y (up) motion is important to the walk signature but also allowing that energy to be repeated on every remaining position is somewhat redundant. So for this reason we choose to extract global translation and treat it separately. This is easiest done by removing the three translation channels for the root bone before conversion to position.

Root rotation is also somewhat independent of the underlying class of action. For the types of motion we are interested in analysing, mostly upright and happening in a standing position, the yaw or rotation on the x-z (ground) plane is irrelevant. The tilting axes, the non-yaw rotations of the character do tend to be important though and we keep them. For more extreme motions like break dancing, acrobatics or judo the yaw is also very important again, but we are not typically looking at those sorts of motions. We remove the yaw rotation after we have calculated the rotation matrix for the root bone, but before any further bone rotation matrices are calculated.

$$\begin{aligned}
X &:= [1,0,0] * R_{root}; X_2 := 0; X := \frac{X}{\parallel X \parallel} \\
Y &:= [0,1,0] \\
Z &:= X \times Y \\
Yaw &:= \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \\
R_{root} &:= R_{root} * Yaw^{-1}
\end{aligned}$$

The above calculates a vector of unit length aligned with X axis of R_{root}, the root bone rotation. It then projects this onto the x-z plane by making the y component equal to 0. This vector is normalised and stored as the new X axis. The Y vector is then assumed to be the cartesian Y axis and the Z vector is the cross product of X and Y. The three vectors are then taken to be the three rows of a rotation matrix. This matrix, Yaw, should represent the Y axis rotation in R_{root}. To remove it we multiply R_{root} by the inverse of Yaw and store Yaw separately for later reconstruction.

Should R_{root} have its X axis pointing up then the approach would fail. There is also failure in the case that the character is upside down (the rotation will be off by 180 degrees). All of the motions files used, bar one, are successfully aligned using this method. It should also be noted that simply removing the Euler Y channel does not work as a means to remove the yaw rotation, Euler angles are not independent.

The results when converted to aligned positions are similar to global positions, but the character's pelvis is centred at the origin and facing in the positive global X axis direction. The residual position and yaw rotation matrix for each frame are stored elsewhere for further analysis or reconstruction. The number of dimensions used to represent each frame of motion is 183 (similar to global motion) and 3 values for the removed global position and 9 values for the Yaw rotation matrix.

Scaled positions

Scaled positions are simply the aligned positions for each motion multiplied by a single scalar. This value is calculated by roughly estimating a height for its associated skeleton and then finding the ratio between that height and a universal height. Remembering that the skeletal topology and bone names for all the used motion files are the same we can calculate the height by summing the length of the set of bones with the following names {*lhipjoint, lfemur, ltibia, lowerback, upperback, thorax, lowerneck, upperneck, head*}. Given a matrix $X \in R^{Nx279}$ representing Aligned Positions for a motion, a universal height U and a calculated height β we perform the following:

$$X = X * \frac{U}{\beta}$$

For reconstruction we must reverse the process. The number of dimensions used to represent each frame of motion is 183 (as normal), 3 values for the removed global position, 9 values for the Yaw rotation matrix. In addition there are F height values, one per motion file.

Normalised positions

Normalised positions are the same as aligned positions but with the positions altered such that the length of the bones are the same for all the animations. A universal set of bone lengths is chosen, currently the universal bone lengths are defined by bone lengths from one arbitrarily chosen ASF file in the database. A better choice might involve choosing values based on the golden ratio. It may also make sense to consider splitting motion databases by male, female, child etc and choosing the nearest set of bone lengths that match the motion under consideration. This is because the proportions of adults, children, males, females can be rather different and thus the motions associated with them different too.

The value we are modifying is V in the previous equation for converting from channels to positions. Given a set of universal bone lengths, $U_i, i \in (1..J)$ and a set of motion bone lengths $B_i, i \in (1..J)$ where J is the number of joints we calculate two arrays per motion, $Norm$ and $Denorm$ as follows:

$$\begin{aligned} Norm_i &= \frac{U_i}{B_i} \forall i \in (1..J) \\ Denorm_i &= \frac{B_i}{U_i} \forall i \in (1..J) \end{aligned}$$

We don't normalise the positions when converting from channels to positions, but instead do it afterwards. This is because there was a need to write a function to undo the normalisation and it is the same as the function for normalising it. Given $Norm$ we look at every frame of motion converted to aligned positions, that is we take one row of $X \in R^{Nx183}$ from before to get a vector $F \in R^{1x183}$ representing that frame of motion. Within a frame the position and virtual markers are split, with $F(1 : 93)$ representing the positions of the 31 joints in a row. Because we don't translate the virtual markers to the ends of the bones, there is no need to scale or modify them for the normalisation purposes, they represent part of a rotation matrix. The position on the other hand must be scaled by the associated joint value $Norm_i$. The effects of the scaling do not stop there, all children must be offset by the change in end bone position of a parent. Rather than cascading the values down the hierarchy each time a change is made to a bone length, traversing the hierarchy multiple times, we accumulate the changes from parent scale changes and pass them onto children in one traversal. To do the reverse, to de-normalise the motion, we apply the same process of changing each frame of motion, but with the array of scale values represented by $Denorm$. The number of dimensions used to represent each frame of motion is 183, 3 values for the removed global position, 9 values removed for the Yaw rotation matrix. In addition there are F arrays of both $Norm$ and $Denorm$ where F equals the number of motion files and the length of each array is J, equal to the number of Joints.

3.2.2 Compression techniques

In this section we discuss the background and implementation of the compression techniques and recognition technique.

Discrete fourier transform

The Discrete Fourier Transform, DFT, is a discretised version of the Fourier Transform in that it converts a sampled signal of fixed length from the time domain, such as the time varying motion channels, into the frequency domain as a fixed set of frequency values. The frequency values can represent an infinite signal in the time domain periodically repeating at the defined frequency.

$$X_n = \sum_{k=0}^{N-1} x_k \exp(^{\frac{2\pi ikn}{N}})$$

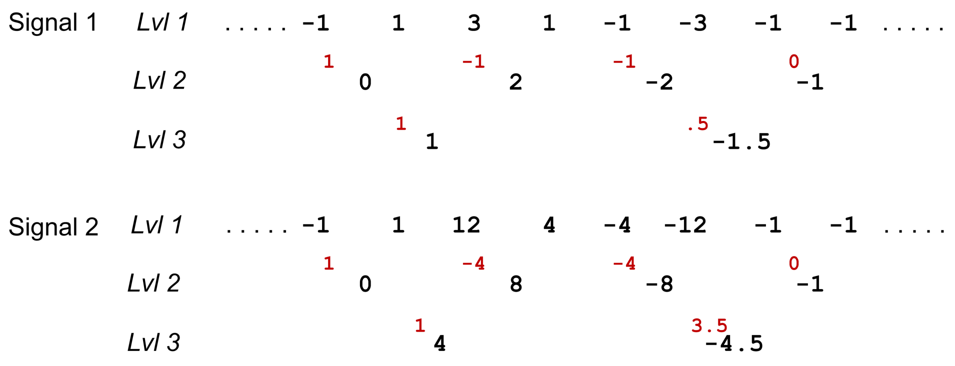
The process can be inverted such that the waves are summed to create the signal that would exist for a given section in the time domain. The inputs can be real numbers, or imaginary. If only real inputs are used, then only real outputs are produced in the frequency domain. This is called the Discrete Cosine Transform DCT because it only uses cosine functions and ignores the imaginary sine based aspect of the DFT.

The DCT also provides for a more compact representation of a time varying signal. Practically speaking when analysing the frequency magnitude of the output of a DFT in Matlab the results mirror or repeat, while with the DCT a similar signal without repeating (and with double the frequency 'bins') can be observed.

Discrete wavelet transform

The Discrete Wavelet Transform is similar to the Discrete Fourier Transform in that it transforms data from the time domain into another domain from which the process can then be reversed. The transformation for wavelets is not as simple as a sum of sine and cosine functions. Instead we have one function, the mother wavelet, that is not periodic over a large time frame. It grows and diminishes over a small localised time interval. This wavelet is then scaled and translated to apply it at different locations and sub-lengths of the time varying signal. The discrete in DWT comes from choosing a discrete set of a positions and scales for the wavelet hierarchically (in a pyramid fashion) to some sampled time varying values.

Not any function is sensibly capable of becoming a mother wavelet due to the requirements for combining the wavelets, but there are still infinite possibilities. The main advantages of using a wavelet based compression, from a compression view point, are the locality of representation in dealing with sharp changes or discontinuities of the input signal.



Signal 1
Lvl 1 ..... -1 1 3 1 -1 -3 -1 -1 .....
Lvl 2 1 0 -1 2 -1 -2 0 -1
Lvl 3 1 1 .5 -1.5
Signal 2
Lvl 1 ..... -1 1 12 4 -4 -12 -1 -1 .....
Lvl 2 1 0 -4 8 -4 -8 0 -1
Lvl 3 1 4 3.5 -4.5

Figure 3.3: Three level Harr Decomposition of two similar signals with approximation and coefficients (offset in red)

Fig 3.3 shows the decomposition of two sampled signals using a Harr mother wavelet. At each level the previous level is approximated using half as many approximation coefficients and an associated detail coefficient. Note that the second signal is similar to the first except that the third to sixth samples have been scaled by four, creating a sharp change in the signal. The Harr transform quickly absorbs these changes locally in its representation. Decomposed transformed coefficients representing the signal further to the left or right of the signal would not be affected. The fourier transform cannot localise the effects in such a way. For animation we generally want a better representation than the Harr wavelet, the Harr wavelet requires many coefficients to represent the mostly smoothly changing nature of motion data.

Principal components analysis

Principal Components Analysis (PCA), is a technique used to reduce data of high dimensionality to that of a lower, more representative dimensionality. Often vectors of high dimensionality are not uniformly distributed throughout the high dimensional space, instead they are aligned along certain hyper planes of lesser dimensions. A visual example, in a 3d space, is that of a thin spiral galaxy made up of many stars. The principal dimensionality of the galaxy is 2d, but existing in a 3d dimensional space. We can imagine that it might be possible to centre a point in the middle of the galaxy, the mean position of a star, and align two orthogonal unit 3d vectors like spokes on the "wheel" the galaxy mimics. Then we could describe the position of each star in the galaxy using the mean and a projection on to the plane defined by the orthogonal unit vectors. PCA helps find the orthogonal vectors of a set of vectors and the importance each vector has in terms of representing the data.

$$x_c = x - E(X) \tag{3.2}$$

$$svd(X_c) = USV^T \tag{3.3}$$

$$Y = X_c V \tag{3.4}$$

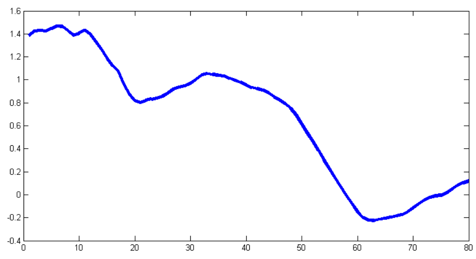
Singular Value Decomposition (SVD) is a decomposition of a square or non-square matrix based on the eigenvalues of the matrix. Equation (3.4) shows the factorisation of a matrix X_c into three matrices U, S and V. If matrix X_c is $R^{N \times M}$, then U is $R^{N \times N}$ and S and V are $R^{M \times M}$. The relationship between X_c and the unit vectors U and V is shown by the eigenvalue decompositions $X_c X_c^T = U(SS)U^T$ and $X_c^T X_c = V(SS)V^T$. S is a diagonal matrix of non negative "singular" values. The matrix V is the one we are most interested in because it turns out to be an array of unit vectors representing the rotation of the M unit cartesian vectors to M unit vectors best aligned with the data in the matrix X_c. The size of the values in S indicate how important each matching vector in V is, in fact $\sum_{j=1}^{M} \sigma_j^2 = \sum_{i=1}^{N} ||x_{ci}||^2$ where σ_j is a singular value from the diagonal of S and x_{ci} is an observation from X_c. The measures of importance let us define the principal components of the data in X_c. If our singular values are sorted by importance then it is trivial to calculated the total squared error for all observations in

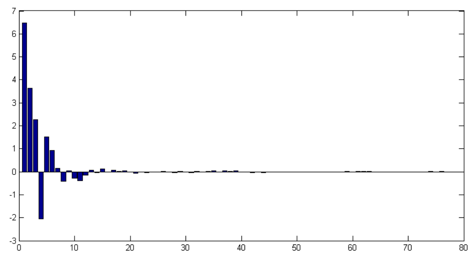
X introduced by projecting using a subset of the vectors in V and reconstructing them.

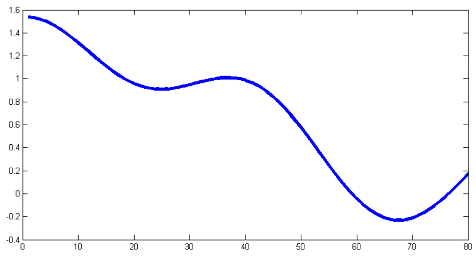
We create X from the motion data. Each row is a frame of motion in the format chosen. While the columns do represent a one dimensional time varying value, the SVD does not take this temporal coherence into account. It simply sees matrix as a collection of vectors that may be correlated via common alignment to a hyperplane. The data is centred by subtracting the mean value of the rows of X, to become X_c, and can then be rotated by the matrix V to create the vectors Y. To reverse the process it is only necessary to transpose the orthogonal matrix V and add the mean to translate the data. The error of reconstruction should be negligible, if all columns of V are utilised, that is if no projection is performed. It is also possible to build the V matrix by taking the covariance matrix of X_c and performing an eigenvalue decomposition upon the square matrix result. The eigenvalues λ of the covariance matrix are the variance of each column of Y and relate to the singular values with the the relationship $\frac{1}{N}\sigma_j^2 = \lambda_j$.

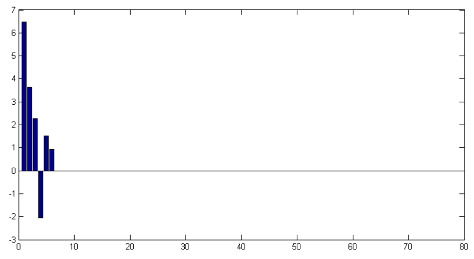
3.2.3 Compression

The compression provided by the above techniques all involve some sort of removal of energy that is considered noise or imperceptible. We are thus dealing with lossy compression. The simple relationship between perception and the values in any of the transformations is that small coefficients are unnoticed by human observers. As discussed before error isn't simply measured as norm based metrics on the frame data, instead the frame data must be further converted to a skinned or animated mesh so that numerical errors relate properly to something in the mesh perceptual domain. For perceptual encoding it would be necessary to relate perceptual problems to the choices of coefficients to alter or remove when dealing with temporal techniques or when dealing with dimensionality reduction techniques such as PCA. The typical approaches for dealing with perceptual issues focus on well known problem areas like foot placement (and other physical constraints) independently of compression for other joints or to perform dynamic inverse kinematic solutions as a fix up afterwards. We don't go into any of these areas - small is removable is the simple approach. There is no entropy encoding and no quantisation of values.


(a) Original signal


(b) DCT using 80 coefficients


(c) Compressed signal


(d) DCT using 6 coefficients

Compression for DCT

We perform the Discrete Cosine Transform on the columns of X or the columns of Y, considering each dimension as a time varying sequence. After the conversion to the frequency domain it is possible to get the magnitude of each discretely measured frequency. There is a relationship between the magnitude and the amount of error that will be introduced to the original time domain signal. So if we choose to remove only the smaller coefficients we can do so without introducing as much error. For many animations, particularly simple periodic animations like walking and running the representation in the frequency domain is compact. Fig 3.2.3 shows the results of removing the smallest coefficients from an example signal of 80 samples. When storing only the largest coefficients it is necessary to also store their indices. 16 bits is enough to cope with animations of up to 9 minutes at 120hz.

Compression for DWT

Compression using Wavelets is also performed upon the columns of X or Y. Again the technique is to look for the largest coefficients and keep only these. There are questions whether to differentiate between detail coefficients and approximation coefficients or coefficients at the various levels/bands of decomposition. Higher band coefficients or approximation coefficients (the coefficients on the highest band) are likely to have a wider effect on the outcome, but then a large detail coefficient at any position must contain important information. The simplest approach of not differentiating between any of coefficients other than by magnitude was chosen. Again indices need to be retained, although the indices now refer to locations on a multi-level decomposition. With a sensible scheme 16 bits can also allow for addressing of coefficients for sequences up to 9 minutes at 120hz.

Compression for PCA

Compression using PCA is performed upon the rows of X_c, a frame of motion with the mean frame values subtracted. The frame is treated as a vector that is then projected onto the principal orthogonal components defined by a sub-matrix V_Q of V where Q is the number of principal components to use in the projection: $y_{proj} = (x - E(X)) * V_Q$. This means we can use $y_{proj} \in R^{1 \times Q}$ instead of $x \in R^{1 \times M}$ for all of the rows of X. Reconstruction of X, $\hat{X}$ can be performed on a row by row basis as $\hat{x} = y_{proj} * V_Q^T + E(X)$. The singular values of the diagonal of S guide us as to how much error is introduced by the projection $\sum_{j=Q+1}^{M} \sigma_j^2 = \sum_{i=1}^{N} ||x_i - \hat{x}_i||^2$ or the variances of the columns of Y found by the decomposition of the covariance matrix can be used. It is typical for 99% of the variance to be accounted for within 10 to 20 dimensions for a given set of similar animations. We compress by choosing an appropriate number of dimensions for the motion.

Using PCA comes at a cost in that the matrix V_Q and the vector $E(X)$, the mean, must be stored as well as the coefficients Y_{proj}. The number of dimensions that need to be retained to reconstruct X with low error rises if X has low joint correlation i.e. if the motion is varied and the vectors start dispersing in their locations in the high dimensional space. PCA is best with multivariate Guassian-like distributions. If this is not the case, as one might expect in a motion capture database, then it would be necessary to use multiple PCA performed upon multiple clusters of points, each PCA requiring a new mean vector and set of projection vectors V_{Qi}. Clearly there is a trade off between splitting the cluster to reduce the number of dimensions to keep, and the cost of setting up a new set of vectors. For this project PCA was performed upon different classes of motion as defined by annotations and human observation. It is not uncommon to split the motions such that the inherent dimensionality of the motions as defined by observing the motion through PCA stays low [23], [21].

Combining techniques

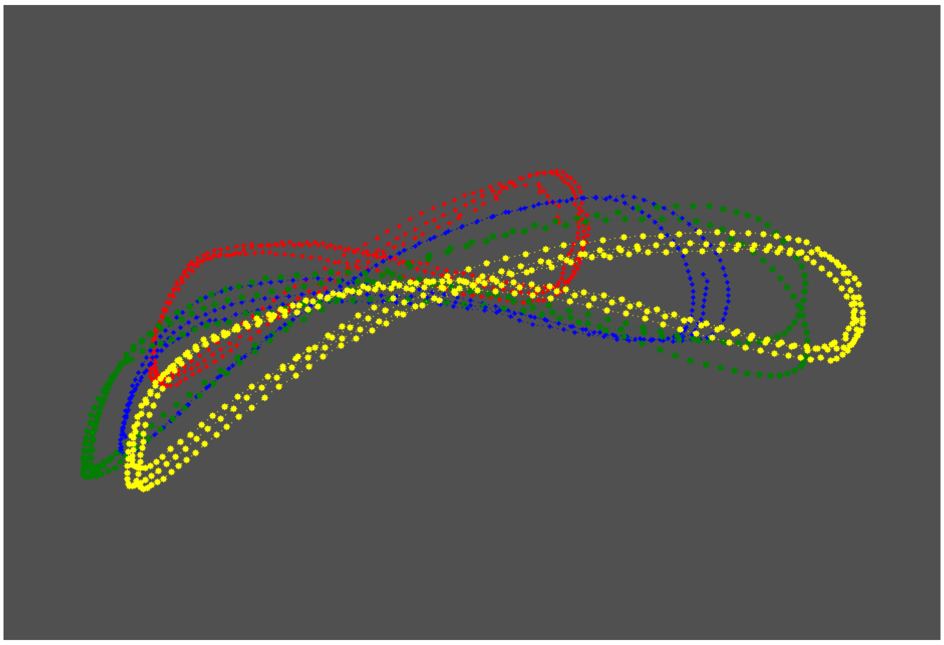


Figure 3.4: First three dimensions of vectors of four walk animations rotated by PCA of the data. Note that the points still follow a trajectory.

If compression is the act of removing correlations from data, PCA helps remove correlations between bone configurations while DCT and DWT help to remove temporal correlations. It doesn't make sense to consider compressed representations of the data after a DCT or DWT as high dimensional vectors, but it does make sense to consider the output of compression via dimensionality reduction as inputs to a DFT or DWT. Simply put, we perform compression on the time varying columns of Y_{proj}. Fig 3.4 shows the alignment of values of four walking motions for the first three dimensions of their projection via PCA. There is still clearly space for compression on the points over time.

Root finding

For the DCT and DWT there is no, known to this author, direct correlation between the coefficients removed and the error of the approximated motions created. For PCA the sum of the squares of the norm of projection errors is quickly found but the skin error is not directly determined from this. So for all techniques if we wish to indicate that a given error is allowable, which is useful from a tools or results checking point of view, then we must use a way of finding for each technique how we can allocate a given amount of error to its compression routine. We do this via root finding for the given error. For the DCT and DWT we do this by algorithmically by bounding two values left and right of the target error, left representing no coefficients removed and right representing all coefficients removed and iteratively chop or segment and test the error introduced by choosing a number of coefficients half way between both extremes. When an error value is found we test it against the desired error, relocating left of right to the previously tested value depending on the current direction of the error. The assumption is that our values are continuously increasing or decreasing when we add or remove coefficients based on magnitude. This is mostly true but there are stepping effects in the progression of error in reality because the removal of a coefficient or not is a discrete choice. We halt when the left and right values have no gaps between them. Convergence is done in $log(N)$ steps with regards to N the number of samples.

A similar process is done when searching for the correct number of dimensions to retain given an error for PCA. The effect of removing dimensions can involve much smaller and then much larger steps due to the quickly changing nature of the variance in PCA. It is important to make sure in all cases that the coefficients retained or dimensions retained errs of the side of introducing less error. In the case where we combine techniques such that PCA is performed first, we can pass the remaining excess error allowance for PCA as an extra allowance for the compression process on the columns of Y_{proj}. All techniques must, when testing the error, revert the process and reconstruct the frames for the entire motion, skin the mesh for each frame and average the norm of the difference of the mesh vertices taken a vector. Due to the poor performance of attempting to skin a mesh within Matlab the routine was written in C++ as a .mex file and also the motion was sub-sampled to 30hz or one quarter the number of frames.

Typically a user of a compression process does not wish to tweak separate values for differing techniques and would instead prefer to apply one value that is considered qualitatively good in general. This is a minimisation problem. A one dimensional routine that splits the error between the two techniques PCA and time varying compression was developed based on the "Golden Section Search". Unfortunately the function being minimised involves performing the rooting finding routines from above many times which are already taking minutes for some data formats. Also with the introduction of aligning motion a new input value for the minimisation routine was introduced making the the problem a more complicated gradient descent problem. While it would be a good area of research (finding the ratios at which the trade off between error for PCA and error for temporal techniques happens) because of time constraints this approach had to be dropped.

3.2.4 Recognition

For recognition we would like to take a motion and identify it as belonging to some class of motion quickly and robustly. From before we saw how one might train a set of poses to represent a centred lower dimensional space and, depending upon the data, the distribution should be gaussian in spread along principal components. One means of classifying a pose as belong to a distribution is the Gaussian density function, equations 3.6 and 3.6 show the one dimensional and multivariate distributions. σ represents the standard deviation of $X_c \in R^{N\times 1}$, while Σ is the covariance matrix of $X_c \in R^{N\times M}$ and x_c is a pose with the mean, $E(X)$, subtracted.

$$P(x|\Omega) = \frac{\exp(-\frac{1}{2}\frac{x_c^2}{\sigma^2})}{(2\pi)^{\frac{1}{2}}\sigma} \tag{3.5}$$

$$P(x|\Omega) = \frac{\exp[-\frac{1}{2}x_c\Sigma^{-1}x_c^T]}{(2\pi)^{\frac{N}{2}}|\Sigma|^{\frac{1}{2}}} \tag{3.6}$$

Our poses are not uniformly distributed throughout the entirety of the M dimensional space and it's for this reason that we can reduce the dimensionality of the space to Q dimensions. But it is also for this reason that attempting to find Σ^{-1} is problematic due to singularities. We might choose then to model our multivariate probability using just the projected data, getting Σ from the covariance of $Y_{proj} \in R^{N\times Q}$. This will correctly classify poses from the testing set and from similar motions. Unfortunately it will also project many poses that are far away, in the Euclidean norm sense, but residing within variance of the Q principal components. That is, some of the $M - Q$ principal components are still very different and we should be rejecting the pose for that reason. It is therefore important to stay in M dimensional space.

Probabilistic principal components analysis

[31] provide a good explanation of how to implement Probabilistic Principal Component Analysis [32]. We provide an overview. Following on from the previous section where we cannot simply model using just the projected values $Y_{proj} \in R^{N\times Q}$ and the values for $X_c \in R^{N\times M}$ are unstable, we might like to break the problem into two parts. First is the probability of being within variance of the projected space and second is being within variance of the thin area of "noise" for all the remaining principal components. When we are using PCA we are dealing with this concept of greater and lesser importance of dimensions (lesser dimensions being "noise"), whereas when we are dealing with poses from X this is not the case. So we would like use all the dimensions of Y. Using the "Mahalanobis" distance as a starting point we can show how to move from X_c to Y.

$$d(x) = x_c\Sigma^{-1}x_c^T \tag{3.7}$$
$$d(x) = x_c[V\Lambda^{-1}V^T]x_c^T \tag{3.8}$$
$$d(x) = y\Lambda^{-1}y^T \tag{3.9}$$

Equation 3.8 shows the Mahalanobis distance and should be recognisable from equation 3.6. The covariance matrix Σ of X_c, when decomposed by an eigenvector decomposition, produces the eigenvectors $V \in R^{M\times M}$ which work much like V from the SVD decomposition of the data matrix X. Also produced is $\Lambda \in R^{M\times M}$ a diagonal matrix of values λ that represent the variance associated with each column of V. This is related to σ as mentioned before. We now have the opportunity to exchange x_c for its rotated form as y, the pose in a space where every dimension has been decorrelated and aligned by principal components. Equation 3.9 is then simply the sum of squares of each variable of y divided by a variance $d(x) = \sum_{i=1}^{M} \frac{y_i^2}{\lambda_i}$. Unfortunately we still have the problem of having very small values for λ in the not so principal components.

At this point a choice is made to segment the components between the Q principal components used for projecting onto and the remaining noisy components related to the error of projection. When this is done our Mahalanobis distance looks like equation 3.10.

$$d(x) = \sum_{i=1}^{Q} \frac{y_i^2}{\lambda_i} + \sum_{i=Q+1}^{M} \frac{y_i^2}{\lambda_i} \tag{3.10}$$

$\sum_{i=Q+1}^{M} y_i^2$ happens to be a known entity, the square of the L_2 norm of x_c minus $\sum_{i=1}^{Q} y_i^2$ or the square of the L_2 norm of $x - \hat{x}$

$$\epsilon(x)^2 = ||x_c||^2 - \sum_{i=1}^{Q} y_i^2 \tag{3.11}$$

This value is divided by the average λ_i associated with the noisy components or $\rho = \frac{1}{M-Q} \sum_{i=Q+1}^{M} \lambda_i$. This approximation is considered to the be optimal estimate. The Mahalanobis distance is now the sum of the square distance within the projected space scaled by the variances of each dimension and the the square error of projection divided by the average variance of all the remaining noisy variances, reducing the remaining dimensions to a one dimensional problem. This gives us some measure of how far a point is away from the projected space in a way that doesn't directly involve knowing the data, but instead working with it in terms of its variances. We reintroduce the results of considering the mahalanobis distance back into the probability function to get equation 3.13.

$$d(\hat{x}) \quad = \quad \sum_{i=1}^{Q} \frac{y_i^2}{\lambda_i} + \frac{1}{\rho} \epsilon(x)^2 \tag{3.12}$$

$$\hat{P}(x|\Omega) = \frac{\exp(-\frac{1}{2}\sum_{i=1}^{Q}\frac{y_i^2}{\lambda_i})}{(2\pi)^{\frac{Q}{2}}\prod_{i=1}^{Q}\lambda_i^{\frac{1}{2}}}.\frac{\exp(-\frac{\epsilon^2(x)}{2\rho})}{(2\pi\rho)^{\frac{(M-Q)}{2}}} \tag{3.13}$$

This formula is then the basis for determining if a pose is within the PCA space generated by X and the first Q dimensions. A class is generated from the training samples that make up X and the pose can come from any frame of any motion file.

3.3 Summary

In this chapter we covered the design and implementation of the code we used to generate the tests. A lot of effort was put into managing the nature of the data, from ASF/AMC to channels, onward to normalised positions and the skinning of a character. Also we covered the techniques for compression and the maths behind probabilistic PCA. In the next chapter we provide the results of those tests and consider what they might mean.

Chapter 4

Evaluation

In this chapter we present the details of the tests and their results.

4.1 Results

All the motion files used originated from the CMU database [29]. All characters use the same skeletal topology, same number of degrees of freedom and only motion files sampled at $120Hz$ were used. Bone lengths and initial pose configurations (bone orientations) may differ between characters. No timing data was retained, the speed of compression could certainly improve in various cases but we wanted to avoid approximations if possible and there was only time for essential optimisations. For the tables in this chapter the abbreviations R, P and T are used to represent input parameters to the compression routines. These are error allowances for the root motion compression, R, PCA compression, P and temporal compression T. The error relates to the average error defined in section 3.1.1.

Table 4.1 shows compression results for different formats and different sets of motions using the DCT technique only on the motion data. Values are average bytes per frame. Table 4.2 shows the same results when the PCA is used to aid compression as described by the combined approach, section 3.2.3. The error allowance is split half between the PCA technique and the DCT technique. Values are bytes per frame once compressed and do not include the cost of compressing the root motion.

Table 4.1: No PCA used. [R:0.0, P:0.0, T:1.0 (DCT)]

	Chan	**Pos**	**ScPos**	**NrmPos**
1 Run Motion (173)	83.39	89.39	89.38	89.39
1 Walk Motion (316)	96.00	57.73	57.73	58.78
4 Run Motions (623)	98.02	109.54	109.50	109.34
3 Walk Motions (1127)	105.40	57.85	57.87	59.06
1 Boxing Motion (4840)	91.01	108.75	108.75	109.08

Table 4.2: PCA is used. [R:0.0, P:0.5, T:0.5 (DCT)]

	Chan	**Pos**	**ScPos**	**NrmPos**
1 Run Motion (173)	123.16	136.89	136.89	136.89
1 Walk Motion (316)	126.96	86.22	86.35	86.00
4 Run Motions (623)	142.21	104.25	104.01	104.95
3 Walk Motions (1127)	147.09	56.40	55.08	56.63
1 Boxing Motion (4840)	99.23	52.28	52.28	52.94

What we see is that making use of the PCA technique is generally a preferable thing to do once a significant amount of frames are compressed. Because we generate the principal component vectors based on the frames in the sets of motions the PCA space represents the frames well at a lower dimension, but the vectors themselves can take up a significant amount of space. The more frames added, the more dimensions needed to reconstruct the motions within an error tolerance, but the increase in numbers of dimensions needed is much slower than that of the increase in numbers of frames. This would suggest PCA comes into its own when databases have many poses clustered around the same values and is weaker when the database has a more uniform distribution of poses.

We note that the ability of the DCT compression is very good for walks particularly when represented in positional format and PCA only starts to beat it for positions when we see multiple files. Runs stand out as performing poorly with either PCA or no PCA and we conclude this is because the run motion clips are short, not giving the DCT compressor a chance to gain ground on periodic motion or in repetition of similar poses for PCA. The clips may often contain only a few footsteps. We blame this on the small size of motion capture labs which don't allow for longer "running in a line" clips. It's expected that should the running motion clips have contained the same number of run cycles as the walk clips that we would see better compression.

Between the different positional formats there is not much differentiation. It seems for compression that attempts to normalise the positions in relation to the skeletal structure of the character does not have a big effect, this matches our expectations from analysing the proportional variance of the different formats with different motion files. We would recommend positional or the scaled positional formats if one were choosing, since these are fast to decompress and give slightly better results than normalising the motions.

Using channels as a format for compression sometimes makes sense, but only if the motion clips are small and there a few of them. When they are small we see the effect of the temporal compression performing poorly as described for the run motions. This makes the positional format, which is almost three times as large as the channel format, retain too many coefficients. At the same time because there are only a few short clips in total, the benefits of PCA are shadowed by the overhead of the principal component vectors and mean. As the number of frames increases the benefits of PCA start to appear. The most interesting point is that PCA never makes sense in the examples tested when dealing with channels. Again this is in line with expectations from studying the proportional variance of the formats, channels require many more dimensions to represent the inherent dimensionality of the numbers they represent, and the numbers they represent are non-linear in the first place.

Table 4.3: Fourier and Wavelet Comparison. [R:0.1, P:0.2, T:0.2]

	Fourier	**Wavelet**
2 Walk Motions (772)	119.06	142.97
4 Walk Motions (1447)	105.61	136.08
1 Run Motion (173)	210.84	235.86
4 Run Motions (623)	188.74	253.84
2 Boxing Motions (10434)	86.77	96.64
1 Break Dancing (770)	256.12	158.84

Table 4.3 shows compression results using the positional data format. The compression technique is first PCA and then either a DCT or DWT performed upon the

remaining time series data. Values are average bytes per frame. The Discrete Fourier Transform as a general compression technique works rather well. This is not entirely surprising in the case of walking or running motions where we expect the periodic nature of the motions to dominate. For the boxing motion the result in favour of the DCT is unexpected. It had been anticipated that due to the sharp punches and random "butterfly" like swings of the upper body that wavelets and their ability to localise in time their transformation would be a better option. But the motion must still be well represented in the frequency domain.

Because of the boxing case a direct attempt to find a scenario were wavelets would perform better was initiated. It turned out to be an animation, with a dancer spinning on their back. The interesting thing about the motion is that it violates some assumptions previously used for extracting the root motion, namely that the person would not be upside down. This means that the code (see section 3.2.1) for aligning the motion generates a yaw matrix, offset incorrectly by 180, and forces all the motion capture data to suddenly change to an orientation 180 degrees opposite. This discontinuity creates the scenario better made for the DWT. The DCT finds it very hard to emulate a discontinuity without using a lot of coefficients. If one were going to perform compression on a database known to contain discontinuities or if it were necessary to perform temporal compression across the boundaries of motion clips we suggest that DWT may be a good choice, otherwise DCT is a good choice.

Table 4.4: High and low error comparisons. [High R:0.5, P:1.0, T:1.0 (DCT)] [Low R:0.1, P:0.2, T:0.2 (DCT)]

	High Error	**Low Error**
29 Walk Motions (10453)	29.47	5.57
2 Boxing Motions (10434)	20.57	5.78
4 Walk Motions (1447)	16.11	4.68
2 Walk Motions (772)	11.89	3.84
4 Run Motions (623)	6.76	2.53
1 Break Dancing (770)	5.23	1.92
1Run Motion (173)	4.47	2.04

Table 4.4 shows a comparison between compression ratios for a higher error al-

lowance and a small error allowance. Low error is a total allowance of 0.5 error while high error is 2.5, spread amongst the three compression parameters. The compression technique is the combined PCA technique using DCT compression. The ratios show how much smaller the data is its final compressed state to the original channel data that described it.

Best compression is achieved for walking motions. This is most likely because we have many clips for which the poses are rather similar to each other, within the motion and with poses of other walking motions. Also the DCT compression works well for the periodic motion as mentioned before. The two boxing motions on the other hand contain a set of poses which require more dimensions to represent and even though roughly the same number of frames are used, the compression ratio is not as good when using high error. In general the less frames the less effective the PCA technique but if lower error is required the beneficial effect of using more frames is minimised. We see this as the ratio between best and worst compression is around about 6 for high error and 2.5 for low error.

We made K classes of motion, from training motion files as chosen by human observation or database annotations (there are search by string facilities on-line for the CMU database). These were used to train a set of principal components, one per class Ω_k, also choosing a separate value Q_k for each such that 99% of the variance was contained within the Q principal components, V^{Q_k}. At the same time we also calculated ρ_k, Λ_k and $E(X_k)$ for the class. Then different test motion files were utilised. Each pose in the motion was transformed by the PCA and reconstructed such that the necessary values of y_{proj} and $\epsilon(x)$ were calculated. This allowed us to use the density function. The values returned from using the PPCA density function are often very large or very small. To overcome this we used the log probability to help analysis of a given pose. Deciding whether or not a pose was within a given class or not is a thresholding problem. We stored the percentage of poses within a motion that were above the threshold.

The results show the percentage of poses from within a set of test motion files that were larger than the threshold value. Several walk files, several run files and two boxing files make up the test motions. Only the root motion aligned version of the data for-

Table 4.5: Class recognition for Positions. [Threshold: -319]

	Test walks	**Test run**	**Test boxing**
Walk class	100.00	0.00	3.07
Run class	0.00	76.25	36.43
Boxing class	0.27	0.00	84.34

Table 4.6: Class recognition for Channels. [Threshold: -353]

	Test walks	**Test run**	**Test boxing**
Walk class	100.00	0.00	38.11
Run class	0.00	50.36	27.14
Boxing class	0.00	0.00	84.61

Table 4.7: Class recognition for Scaled Positions. [Threshold: 278]

	Test walks	**Test run**	**Test boxing**
Walk class	100.00	0.00	2.20
Run class	0.00	77.86	35.54
Boxing class	0.31	0.00	84.69

Table 4.8: Class recognition for Normalised Positions. [Threshold: -265]

	Test walks	**Test run**	**Test boxing**
Walk class	99.68	0.00	0.87
Run class	0.00	91.96	26.61
Boxing class	0.37	0.00	80.53

mats were used due to the more compact nature of the representation. The threshold value was determined for each file format such that 300% of poses tested had matched a class between the nine combinations. Just using the one threshold doesn't take into account the different nature of each data format. For example scaled positions have a higher natural threshold because the data is scaled down by a factor of the typical height of a skeleton before processing.

While this is just a simple test, visually inspecting the graphs seems to suggest that all formats are capable of some reasonable form of class detection given the chosen motions for training and testing. The channel format does not perform as well as the positional based representations with significant misclassification happening for the boxing motions onto the walking and running class and poor classification of run onto the run class. The positional representations prove that the test data for run could be strongly identified as being in the run class although they also struggle false positives for test boxing data with regards to the running class. The normalised position seems to perform the best, getting the most percentage on the diagonal and less on the upper and lower triangles. This is more impressive when we consider the scale of the data for normalised positions is similar to that of the positional representation, but it uses a higher threshold value. So for classification we would consider using normalised positions.

Chapter 5

Conclusion and future work

In chapter one and through out the dissertation we focused on answering several questions and the results section 4.1 highlights our answers. To the question of using PCA for compression we discovered that with sufficient data to compress and choosing data representations that require less dimensions for reconstruction PCA does provide better compression. Aligned positions are deemed to be a better general representation, channels perform rather poorly, consistently worse in our tests.

It would be fair to say that it would be better to have had more results for this test. As mentioned in 3.2.3, optimising the input error allowance between the three values of root motion error, pca error and temporal compression was a goal and is definitely of interest for future work. Maybe the choice of ratio for PCA to temporal error allowance (half and half each) was punitive for channels and beneficial for positions. More work would need to be done to improve and speed up the algorithms relating to the minimisation of bytes.

For questions of time series data we found that the DCT performed better over all and was only beat when challenged with a significant discontinuity in the values of the motion capture. DCT performed particularly well for periodic motion that got the chance to repeat within a motion clip, such as with walks, but fared poorly for running where the repetitions were less. It also performed well in the case of the boxing. Again more data points would be useful and one obvious technique that would need to be added for comparison is the use of Bezier cubic curves or splines. They have the ability to localise and represent smooth changes to values typical for character motion and have been used as a compression technique for a long time. There are also many different types of wavelet basis, some of which may be better suited for different types of motion than the one chosen universally for the DWT this project. Further work on how to identity the best wavelet basis would make sense. Also for both DCT and DWT further work on optimal selection of coefficients, such as that described in [26] is a good idea. More work on relating the original signal energy to the transformed coefficients would help speed up coefficient selection if such techniques exist.

Looking at overall compression we discovered that as error lowered the benefits of the compression techniques weakened, that is, PCA combined with a DCT temporal technique was less able to model the local space and trajectory of projected poses without spending proportionally more coefficients that those of higher error - even with sufficient frames of motion to help. One obvious improvement here is to segment the motions and attempt to model the segmented motions in lower dimensional spaces. Identifying to which space a segment belongs is a classification problem, which for these tests has been performed manually.

For compression we chose the direction of suggesting an error allowance which the motion must conform to after compression. It may have made more sense, in terms of work, to have attempted a fixed byte approach. With this we would quickly sum up bytes without needing to transform all the motion from it's compressed state to test the current error. From a research point of view though, it is more likely we chose the correct way to go (if more hindered) as the questions related to fixed error are more revealing than those relating to fixed bytes.

Looking at the results of classification we were surprised to find that collections of like-annotated motion are relatively well differentiated for all choices of data representation. Normalised positions seem to provide the best results. This suggests that we might be able to use classification of this manner for quick indexing into a database of motion given pre-trained types.

One application of this technique as future work might be to automatically assign motions to one class or another for compressing a database. E.g. motions are added one at a time, if they match a class within a tolerance, then we know that it resides in the same local space within a set of Gaussian variances. If it does not, we create a new class. When added the motion is not yet compressed, but used to retrain the PPCA data for that class (principal components, mean, etc). This is done for all motions and then repeated until convergence (no motions change class). This is similar in approach to k-means for data points, but testing for the nearest PCA space instead of mean. Once in a space, all motions are considered to be of the same motion type so optimisation particular to that motion class should make sense. For example a specific ratio of error allowances could be minimised and tests for different types of temporal compression tried to learn what is best for that class using just one or two motions from the class. The results of such an application would be a good case study of all the data in a motion capture database, highlighting preferences per motion class and the sorts of local spaces that each motion matches to.

As a self-assessment of the research done, it is clear that more results are needed for compression and an application like the above would have provided the real test of classification and compression of the entire database. Having said this, for the given time frame, I'd like to think some useful results arose and some useful implementation details were outlined that may be of interest to others in future.

Bibliography

[1] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, (San Antonio, Texas), pp. 473–482, ACM, 2002.

[2] P. Glardon, R. Boulic, and D. Thalmann, "Pca-based walking engine using motion capture data," in *Proceedings of the Computer Graphics International*, pp. 292–298, IEEE Computer Society, 2004.

[3] O. Arikan, D. A. Forsyth, and J. F. O'Brien, "Motion synthesis from annotations," *ACM Trans. Graph.*, vol. 22, pp. 402–408, 2003.

[4] J. Assa, Y. Caspi, and D. Cohen-Or, "Action synopsis: pose selection and illustration," in *ACM SIGGRAPH 2005 Papers*, (Los Angeles, California), pp. 667–676, ACM, 2005.

[5] G. Liu, J. Zhang, W. Wang, and L. McMillan, "Human motion estimation from a reduced marker set," in *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, (Redwood City, California), pp. 35–42, ACM, 2006.

[6] E. Keogh, T. Palpanas, V. B. Zordan, D. Gunopulos, and M. Cardle, "Indexing large human-motion databases," in *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, (Toronto, Canada), pp. 780–791, VLDB Endowment, 2004.

[7] A. Bruderlin and L. Williams, "Motion signal processing," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 97–104, ACM, 1995.

[8] A. Witkin and Z. Popovic, "Motion warping," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pp. 105–108, ACM, 1995.

[9] E. Keogh, "Exact indexing of dynamic time warping," in *Proceedings of the 28th international conference on Very Large Data Bases*, (Hong Kong, China), pp. 406–417, VLDB Endowment, 2002.

[10] L. Kovar and M. Gleicher, "Automated extraction and parameterization of motions in large data sets," *ACM Trans. Graph.*, vol. 23, pp. 559–568, 2004.

[11] G. Liu, J. Zhang, W. Wang, and L. McMillan, "A system for analyzing and indexing human-motion databases," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, (Baltimore, Maryland), pp. 924–926, ACM, 2005.

[12] Y. Lu, I. Cohen, X. S. Zhou, and Q. Tian, "Feature selection using principal feature analysis," in *Proceedings of the 15th international conference on Multimedia*, (Augsburg, Germany), pp. 301–304, ACM, 2007.

[13] K. Forbes and E. Fiume, "An efficient search algorithm for motion data using weighted pca," in *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, (Los Angeles, California), pp. 67–76, ACM, 2005.

[14] Y. Lin, "Efficient human motion retrieval in large databases," in *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, (Kuala Lumpur, Malaysia), pp. 31–37, ACM, 2006.

[15] M. Mller, T. Rder, and M. Clausen, "Efficient content-based retrieval of motion capture data," *ACM Trans. Graph.*, vol. 24, pp. 677–685, 2005.

[16] M. Mller and T. Rder, "Motion templates for automatic classification and retrieval of motion capture data," in *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, (Vienna, Austria), pp. 137–146, Eurographics Association, 2006.

[17] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," *ACM Trans. Graph.*, vol. 21, pp. 491–500, 2002.

[18] O. Arikan and D. A. Forsyth, "Interactive motion generation from examples," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, (San Antonio, Texas), pp. 483–490, ACM, 2002.

[19] L. Kovar and M. Gleicher, "Flexible automatic motion blending with registration curves," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, (San Diego, California), pp. 214–224, Eurographics Association, 2003.

[20] J. Chai and J. K. Hodgins, "Performance animation from low-dimensional control signals," in *ACM SIGGRAPH 2005 Papers*, (Los Angeles, California), pp. 686–696, ACM, 2005.

[21] J. Barbi, A. Safonova, J.-Y. Pan, C. Faloutsos, J. K. Hodgins, and N. S. Pollard, "Segmenting motion capture data into distinct behaviors," in *Proceedings of Graphics Interface 2004*, (London, Ontario, Canada), pp. 185–194, Canadian Human-Computer Communications Society, 2004.

[22] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, 2001.

[23] G. Liu and L. McMillan, "Segment-based human motion compression," in *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, (Vienna, Austria), pp. 127–135, Eurographics Association, 2006.

[24] O. Arikan, "Compression of motion capture databases," in *ACM SIGGRAPH 2006 Papers*, (Boston, Massachusetts), pp. 890–897, ACM, 2006.

[25] D. Tolani, A. Goswami, and N. I. Badler, "Real-time inverse kinematics techniques for anthropomorphic limbs," *Graph. Models Image Process.*, vol. 62, pp. 353–388, 2000.

[26] P. Beaudoin, P. Poulin, and M. van de Panne, "Adapting wavelet compression to human motion capture clips," in *Proceedings of Graphics Interface 2007*, (Montreal, Canada), pp. 313–318, ACM, 2007.

[27] R. McDonnell, M. Larkin, S. Dobbyn, S. Collins, and C. O'Sullivan, "Clone attack! perception of crowd variety," *ACM Trans. Graph.*, vol. 27, pp. 1–8, 2008.

[28] C.-C. Hsieh, "Motion smoothing using wavelets," *Journal of Intelligent and Robotic Systems*, vol. 35, pp. 157–169, Oct. 2002.

[29] C. M. University, "Cmu motion capture database." Website, 2008. `http://mocap.cs.cmu.edu/`.

[30] N. Lawrence, "Matlab utility toolbox for loading motion capture." Website, 2008. `http://www.cs.man.ac.uk/~neill/mocap/`.

[31] B. Moghaddam and A. Pentland, "Probabilistic visual learning for object representation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 19, pp. 696–710, 1997.

[32] M. Tipping and C. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 61, pp. 611–622, 1999.