Using Mocap Envelopes to Reach Target Points

Alan Robinson

30th August 2001

This dissertation is a
Part Requirement for the
MSc. In Software Systems Technology

Abstract

This work examines the ways in which motion capture (**mocap**) sequences can be mixed together, and produces an interactive game where the figure attempts to catch a ball.

A program is produced which plays back mocap sequences, allowing the operator to vary parameters and mix two sequences together in real time. From this an analysis is made of the data types which can be used for the storage and interpolation of the motion data. Euler angles, transformation matrices, axis/angle pairs and quaternions are used, and their respective advantages and disadvantages are examined. This part of the work concludes with a recipe of issues which need to be addressed before choosing the sequences.

Key points in the figure`s motion (in this case a trace of the movement of the right hand) are stored for each of the mixes as a set of **envelope** data, which therefore describes all possible positions of that point in the range of mocap mixes. This data is calculated at the start of the interactive game program. In the game the user can vary the mix percentages, thus moving between the two sequence motions, in order to find a point where the right hand will coincide with the spatial position of the ball – the **target point**.

This simple game is then tested by a number of players, and evaluated in terms of its realistic interpretation of motion and sense of the catching of the ball.

Acknowledgments

The author would like to thank the following people:

Dr. Alan Watt
Dr. Steve Maddock
Michael Meredith

of the University of Sheffield Department of Computer Science, for their help throughout this project.

Joe Ludlam
Thomas Ludlam
Susie Robinson
Sheila Robinson

For playing and evaluating the catching game.

Daphne Ingham

For encouragement and help throughout the project.

Contents

List of Figures

Plagiarism Declaration

DECLARATION

All sentences or passages quoted in this thesis from other people's work have been specifically acknowledged by clear cross referencing to author, work and page(s). I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this thesis and the degree examination as a whole.

Name (please use block capitals)

.....

Sign .....

Date .....

Chapter 1:Introduction

1.1. Animation and Motion Capture

Motion capture is essentially the recording, processing and playing back of a representation of a moving object, such as that of the human body. In that sense motion capture has been with us since the beginning of moviemaking. Running alongside the photographic process of making movies has been the work of the animator, who achieves the illusion of movement by manufacturing a succession of images.

The literal translation of "manufacturing" is "making by hand", and the animator performs this manufacture by a combination of drawing, painting and automated production, including the use of computers.

The difference between the photographic and manufactured image, both still and moving, has not always been obvious to the uncritical observer, but the subliminal signs are always there if one looks. Could Canaletto have produced such detailed masterpieces of perspective, light and shade without tracing the image from his *camera obscura*? We instinctively recognise these pictures as work of a photographic type, as we do the roughly rotoscoped drawings in the film *Yellow Submarine.* Similarly, if one pictures a silhouette of a moving person, every muscle in the body and dynamic movement of a piece of clothing makes a contribution to that apparently simple outline. Each subatomic particle plays its part, and it is not surprising that humans, even with the aid of computers, find it impossible to match that level of detail.

Many artists and animators have used photographic and manufactured elements in their work, sometimes trying to hide the differences. Walt Disney in *Snow White* did this very successfully, whereas sixty years later in a similarly ambitious project the BBC in *Walking with Dinosaurs* clumsily superimposed computer generated creatures onto a filmed landscape.

Why do animators try to mix the unmixable? Animators are trying to give life to a lifeless object, and they need all the help they can get to create anything but the simplest forms. In particular, humans know their fellow humans` bodies so well that they immediately recognise an odd or infeasible action in an animation of a person. Analysis of real human movements can help us to create the illusion of motion, but the process of translating the analysis accurately is difficult. If one were simply tracing or rotoscoping each frame of a movie that would be easy enough, but one of the points of animation is to animate fantasy figures which do not exist, and so the artist has to employ techniques to move from the recorded image of, say, a human being to the required motion of a space alien.

One of the main goals of motion capture is in the creation of human personalities. Children are entertained by fantasy characters, whereas adults need their players to be real live humans. *The Magic Roundabout* and *Yogi Bear* personify humans; grown

ups watch soaps, sports events, game shows and docudrama – "all human life is there". Motion capture has the advantage that it begins with real live people, and work is being done to preserve and modify the personality of the original actors; how far this can be achieved will be a major issue in the future.

1.2. The Need for Mocap Merging

In the traditional movie every action is recorded and then edited into a fixed position in the final sequence. However, in interactive media such as computer games it is difficult to predetermine all combinations of actions which may be required. Simple platform games can offer a small selection of actions, but a sports game attempting a degree of realism needs a way of utilising mocaps in an interactive way. This has normally been achieved by using one or two mocaps and translating the motion to reach a desired position. For instance, as a goalkeeper attempts to save the ball in a standard motion sequence, if the player needs to reach higher the whole of the body will be translated upwards. This produces an unsatisfactory and unrealistic motion.

This project overcomes these unsatisfactory translations by giving a range of realistic motions based on the merging of two or more mocaps, as described in section 1.5.

1.3. The Data Model

Motion capture is used to collect a set of data describing the motion of a figure in a short sequence of movements. The data is collected by placing optical or magnetic markers on an actor`s joints and other parts of the body and recording their motion as they perform the action. This data is then stored in a suitable data structure which will properly describe these movements to the required level of detail. The data is then available for processing to adapt the motion in a number of different ways before outputting or playing back the results.

The data model usually employed in motion sequences is based on an articulated skeleton of the figure. A starting point (in a human figure it is the hip) is connected in turn to each joint in the body in a hierarchical structure, e.g. hip to upper leg to lower leg to foot to toe. Each bone is of a fixed length but one end can move in three dimensional space from its connected parent joint in the hierarchy

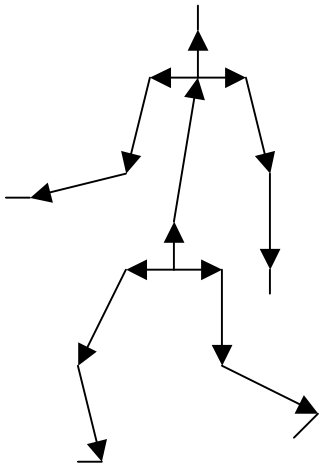


Figure 1.1: Hierarchical model of the human figure. Adapted from Alan Watt (2000) page 503.

The usual way of describing the three-dimensional motion of each bone is by multiplying together three transformation matrices which represent three Euler angle rotations using the parent joint as the origin. The standard file formats such as .bvh and .htr present this data in the form of an offset (x, y, z) and a 3D rotation (ϕ, θ, χ) for each joint.

There are a number of problems associated with this data model, especially when interpolating between different sequences, and it is necessary to analyse in detail the parameterization of the data and how the merging operations will be performed.

1.4. Motion Capture Applications

A detailed look at current research projects in the field of adapting motion capture sequences is presented below in chapter 2. Some of the areas examined are:

- The need to squeeze every ounce of use out of the original motion capture data, which is very expensive to produce
- The requirements of the interactive computer game which demands movements which cannot be predicted by the original capture data
- The creation of fantasy characters and situations
- Editing together different motion capture sequences in a seamless fashion
- Giving the animator control over target positions that may be achieved over the course of the motion
- Analysing the resultant motions and making subjective character judgements – do the characters look heavier, or sadder, as a result of that last operation.

In addition, algorithms employed more fully in the creation of dynamic computer generated figures have been incorporated into work in motion capture transformations.

1.5. Motion Capture Envelopes

This project concentrates on the use of motion capture envelopes. When two original motion capture sequences are merged together, a resultant sequence is produced. If a merged sequence were produced for every possible combination of merge values, the result would be a set of new sequences covering all intermediate motions. This would produce an "envelope" set of motions. A desired action with a specific target location could then be chosen from the envelope set which best matches the required specification. The project will continue work in the University of Sheffield Computer Graphics Research Group to examine the usefulness of envelopes in reaching target specifications.

This project will produce sets of data which represent relationships between a given merge and its resulting action. These stored sets will be searched by the user in a live situation during the game to find the most appropriate action. In practice this means that an envelope of target positions can be searched, and the relevant merge used which reaches the target.

Ideally the envelopes will be used in a simple interactive game in which the figure attempts to catch the ball. At this point some questions come to mind which must be addressed by the project:

a. Do the results provide an acceptable motion for the viewer?
b. Is the `liveliness` still discernible in the modified sequence?
c. Are the operations computationally achievable in real time?
d. Are mocaps available which will produce acceptable results?
e. Can the envelope data be stored in a usable form?
f. Is there a suitable user interface and visualisation?
g. Can the envelope set be used interactively in a game?

The project will begin with an investigation of the motion data types and merging operations, and conclude with the production of the catching game.

Chapter 2:Review of Previous Work

2.1. Collecting Motion Capture Data

The research being done in regard to motion capture covers a number of disparate approaches. This is not surprising in a subject which is attempting to marry numeric data and operations with a subjective view of the results. The review begins with a look at how the raw data is collected (2.1), continues by looking at computer-generated approaches (2.2), and then discusses three current research areas which can deal with motion capture sequences (2.3, 2.4, 2.5).

Motion capture sequences are usually created in a motion capture studio. David Surman (1997)of MediaLab gives a detailed account of the history of motion capture. Optical or magnetic sensors are used for the studio recording process, with optical systems being cheaper and more manageable but suffering from occlusion when one sensor is hidden from view. The breakthrough in the acceptance of motion capture came at SIGGRAPH '93 when Acclaim showed a two-character animation done entirely with motion capture. Their system was able to track up to 100 points simultaneously in real time. Their system is proprietary and used only for their in house games production.

Not surprisingly, work is being carried out by Microsoft Research into motion capture animation, using researchers from many US universities. Bodenheimer, Rose, Rosenthal and Pella are members of the Human Figure Animation Project in the Microsoft Research Graphics Group. In this they give a detailed account of their work with the HFAP in taking human performance sensor data and producing animations of articulated rigid bodies.

Another method of producing motion data is by the analysis of video footage. This subject is being researched by James Davis and Aaron Bobick at the MIT Media Lab and by Norman Badler and others (1999) at the Center for Human Modelling and Simulation, at the University of Pennsylvania. This approach has the advantage of being accessible, less costly, less encumbering for the actors, and working in a wider variety of environments. It would also introduce the possibility of analysing legacy footage, an obviously exciting prospect.

Motion capture sequences are now readily available in standardised file formats, although it appears that most commercial companies still produce data tailored to their specific needs. This suggests that modification of motion sequences is still not trusted by serious commercial producers.

2.2. Producing Computer Generated Motion

The alternative to using motion capture is to produce motion sequences by operations on the hierarchical skeleton. There are two approaches, using kinematic or dynamic methods, which are discussed by Alan Watt (2000). The simple Forward Kinematic method will start at the top of the hierarchy (the hip joint in human figures) and by applying transformations successively at each joint the figure moves. Every mistake in a higher joint thus transfers all the way through the figure, possibly magnifying as it goes. To avoid this problem Inverse Kinematics are now widely used, whereby an end effecter gives a target position for, say, a finger, and the finger then attempts to reach this point in a straight line, but is constrained from doing so by the ripple through of motion to the higher levels.

Dynamic methods of producing motion use physical laws to mimic a real figure more accurately. One of the problems of adapting motion to other figures is that the dynamic performance of different bodies varies. Jessica Hodgins and Nancy Pollard (1997) this problem using the example of a running man, whose motion is adapted to figures of a simulated child and woman. The process is also used to show a figure morphing from one character to another during the motion period. The size, mass and moment of inertia of the synthetic character are first derived from a polygonal representation of the figure, and corresponding parameters are defined for the original and the new characters. Dynamic motions are created using parameters for the torque of each joint and constraints such as ground contact, and the resulting motion is then scaled for the different bodies. A search process relating to control parameters for a specific action (e.g. running, cycling) then finds the optimal solution. This search process is a means by which a precreated set of data can be used interactively in a motion sequence.

Whichever method is used to 'move' the figures, there are then a number of ways in which the motion can be adapted. The most common is by using spacetime constraints.

2.3. Constraining and simplifying the motion

Dynamic modelling is complex to compute, and the situation becomes more difficult when the animator wants to add extra characteristics to the motion. The use of spacetime constraints offers a tool for providing user interaction.

The notion of spacetime constraints was first suggested by Andrew Witkin and Michael Kass (1988). The entire range of the required motion is considered simultaneously, and an algorithm or "solver" is used to calculate the motion, constrained by the user`s requirements of spatial position in relation to time – hence "spacetime constraints". The task is then to provide a solution which minimizes the work and energy expended.

The mathematical problem which needs to be solved is large, and beyond the reach of interactive editing. Michael Gleicher (1997) tackles this problem by outlining a method of user interaction in the ongoing parts of the motion. Although he still users a spacetime problem solver, approximations and omissions are used to simplify the

calculations; he argues that the introduction of user interaction can offset these deficiencies.

In particular he does not include constraints for gravitational laws and $f = ma$, on the grounds that firstly they add considerably to the computation required and secondly that the creator may actually wish to have his or her character flying effortlessly through the air. Secondly, he omits the objective of energy minimization, again on the grounds that it may not be a requirement of the animator.

Another HFAP project, "Efficient Generation of Motion Transition using Spacetime Constraints" allows the user to manipulate, break up, and reassemble motion data.

Simplifying the motion has also been a goal of Popovic and Witkin (1999). The process involves constructing a simplified character model using the original motion, simplifying and reducing the number of degrees of freedom, and finding a spacetime optimized solution to this simplified model. The motion change is then mapped onto the original motion sequence, to produce a new sequence.

The usefulness of this system is that it allows a wider range of dynamic effects to be applied to the model, which is simpler and therefore requires less computational effort. A library of models can be collected to provide the animator with a range of motion types which can be mapped on to the original sequence.

Not surprisingly, this method was found to be best suited for the animation of high-energy, dynamic characters. For slower actions the results were less acceptable. Moreover, it is not possible to combine motions – such as having a person running and waving at the same time – without going back to the original simplified model and making suitable adjustments. Here again we see the same process of simplification of the complex dynamic model, and creation of a library of reusable data.

An obvious use of motion sequences is to make them adaptable to different figure types. It sounds enticingly easy, but in practice subjective viewing of the results shows that various factors beyond simple resizing are critical to their effectiveness. This technique is called 'motion retargetting'.

2.4. Motion Retargetting

Many of the above techniques have involved imposing a motion sequence upon a different figure. Motion retargetting is specifically the process of adapting an existing motion sequence to that of another figure. The new figure must have an identical structure, i.e. the same number of joints connected in the same way, with the same number of degrees of freedom. The new body will have some different segment lengths

Michael Gleicher (1998) outlines the problems encountered in a simple projection of the motion from one figure to another, e.g. the feet may slide along the floor, not

touch the floor at all, a target object may not be reached or an expected collision may not occur. Constraints to make these things happen are added to the motion, and Gleicher suggests meeting these constraints at the expense of less achievable aims, such as preserving the quality of the original motion.

The motion retargetting method finds an initial estimate of the solution by suitably scaling the translation parameters, finding a centre of scaling. choosing a motion-displacement curve and continually solving and feeding back the results until the constraints are satisfied. These constraints include end-effectors, two points following the same path, and the orientation of a vector.

Other work relates to this area of research. Hodgins and Pollard (1997) described an algorithm for automatically adapting existing simulated behaviour to new characters. Litwinowicz (1994) and Bruderlin and Williams (1995) all worked on signal processing in relation to motion. High frequency filtering can be used to smooth out the jerkiness of motion produced by Inverse Kinematics, where for instance the foot may have to suddenly snap to the floor from one frame to the next.

It is difficult to be sure which frequencies are likely to be undesirable, but Gleicher compromises (again) by applying both very-low-pass and very-high-pass filters. This area of signal processing is the subject of much current research.

2.5. Signal processing

Signals representing the movements of joints can be analysed; the hand of a running figure would produce a broadly periodic signal. Work has been carried out by Bruderlin and Williams (1995) and by Unuma *et alia* (1996) in analysing the data provided by motion signals in terms of their frequency, phase and amplitude.

The advantage of this approach is in being able to use a large set of existing mathematical tools, especially in the field of audio and video processing. The authors consider analysis using spline-based representation and periodic representation. Periodic representation allows the use of Fourier analysis, which requires a periodic signal. However, non-periodic elements caused by biomechanical and external factors introduce a noise into the signal which is an important ingredient in the natural-looking and textural appearance of the motion.

Jonas Gomes, Luiz Velho, Fernando Wagner da Silva, Siome Klein Goldenstein (1999) suggest an approach using harmonic decomposition techniques and a variation of the Discrete Cosine Transform, the Lapped Cosine Transform. One result of their research is to discover coupling patterns in the movement of joints, for instance there is a strong dependency between knees, feet elbows and hands, and the motion of the motion of the upper arm and leg joints. There is also a weak dependency between the joints of the arms and the legs.

2.6. Summary of Previous Work

Two separate problems are considered: the animation of a figure using computer processes, and the modification of an existing sequence usually coming from motion capture data. Although these may seem like distinct problems, the research described above suggests that they have common themes in the combination of dynamic modelling, methods of transformation, and human involvement. In practice both approaches progress from observation of real life actions, to a data model of that action, to an attempt to manipulate that action in the desired way, to a subjective judgement of the final motion.

A number of common ideas are expressed in the work under review:

- The original motion sequences are seen as a valuable commodity to be worked and reused as much as possible
- Any modification of the original material is likely to degrade the quality of the motion and make it less appealing to the viewer
- The process should be made as automatic as possible in areas where human involvement would be labour intensive and not creative
- The animator should have as much control as possible over the variations in motion, especially where those variations will contribute to the desired characteristics of the creation.
- The original sequence may have come from live action, whereas the required action may be physically impossible or infeasible.

Much current research is characterised by a pragmatic approach to finding solutions which give the user valuable tools, even if the process is a mixture of different techniques. The work being carried out on interpreting existing action as a data structure moves into an area of thought where all recorded action can be modified, and the boundaries between real and imaginary become more blurred. Mixing archive and live footage is currently popular in adverts, but relies on labour intensive work. Recreating Marilyn or Humphrey using quaternions and spacetime constraints would be very attractive to the media industry, which is happiest when it can employ its favourite personality clichés.

The Human Figure Animation Project has compiled a large collection of motion-related data and research efforts, and they have the advantage of applying their research to commercial environments. This provides a high level of feedback, and much of their work is viewable from the website. The "verb and adverb" project (1999) seems idiosyncratic, but is an interesting attempt to categorize derived actions in a usable way.

The work on signal processing, such as that by Gomes *et alia* (1999)looks interesting but is currently underdeveloped. Certainly when one considers how useful this approach has been in audio analysis and video compression, it would be an attractive research area.

Many authors, including Jessica Hodgins and Nancy Pollard (1997) and Michael Gleicher (1998) have looked at the way that one sequence can be adapted for different types of characters. This is obviously a key area of work if motion capture is going to have any long term uses. Apart from the benefit of reusing existing material, the technique also incorporates the advantages of dynamic simulation with a degree of user control over the final product. The resulting images have a reasonable level of authenticity.

2.7. Sheffield University Computer Graphics Research Group

2.7.1. Michael Meredith

Kinematics and Biomechanics in Adapting Motion Capture Data For Use in Computer Games

Michael Meredith is working on a number of areas related to the use of motion capture sequences. These topics closely match the research areas outlined in the literature review. Although he is not dealing specifically with motion capture envelopes he is using algorithms and other tools which will be relevant to the project.

2.7.2. Michael Jones

Mocap Merging: A Solution for Real-Time Modification of Motion Capture Data in a Games Environment.

This project looked at ways of merging two motion capture sequences in a number of ways defined by the user, who could view the results and make a subjective judgement about the most effective type of merge. A "merge factor" was defined as a proportion of the two mocaps, and a number of profiles were provide – linear and curved – which varied the merge factor as the sequences progressed.

Jones` study achieved a reasonably acceptable resultant motion, except at the beginning and end where a degree of translation was likely, and in some profiles a

"flipping" effect occurred. This appears to have been due to joint angles passing through the lines of origin, and could be avoided by the addition of a suitable constraint.

2.7.3. Stephen Dann

Motion Capture Envelopes: An Investigation into Real-time Modification of Motion Capture Data in a Game Environment

Stephen Dann`s project examines the use of merging to provide a set of new motion sequences which are intermediate to two or more original sequences. The range of these new sequences is described as the "envelope" of possible motions performed when a proportion of one sequence is merged with a proportion of another.

The project uses two motion captures of a goalkeeper catching a football at a particular point in time. The envelope therefore describes a line of catch points as the motion sweeps from the first sequence to the second, defined by the proportion of merging. This set of points can then be used to match the correct sequence against the point at which the football arrives in the plane of the goalkeeper (the goalmouth).

This approach is an alternative to those of retargetting mentioned above, and achieves a reasonable result without great expenditure in computational resources and time. The project does not describe the whole envelope created by the merged motions, and Stephen suggests that this would be one basis for further research.

Chapter 3:Analysis of Motion Data Types

3.1. Data Representation of Motion Sequences

The most common method of representing the motion of a figure uses a hierarchical skeleton structure, which is transformed into a new position in three dimensional space for each frame of the motion. The two file formats used in this project – BioVision`s .bvh format, and Motion Analysis` .htr format, use the system described here.

We can start the construction of the figure with a collection of disjointed bones, each of which is aligned along the Y-axis of its own cordinate system, with the notional "front" of the bone facing the Z-axis. The axes are illustrated for the chest, right foot and right hand. Each bone has two nodes at the points where it is connected to its neighbours or, more precisely, connected to the nodes of its neighbours. The bone's overall length (b), and the offset between a bone's nodes (o) are shown for the right upper leg.

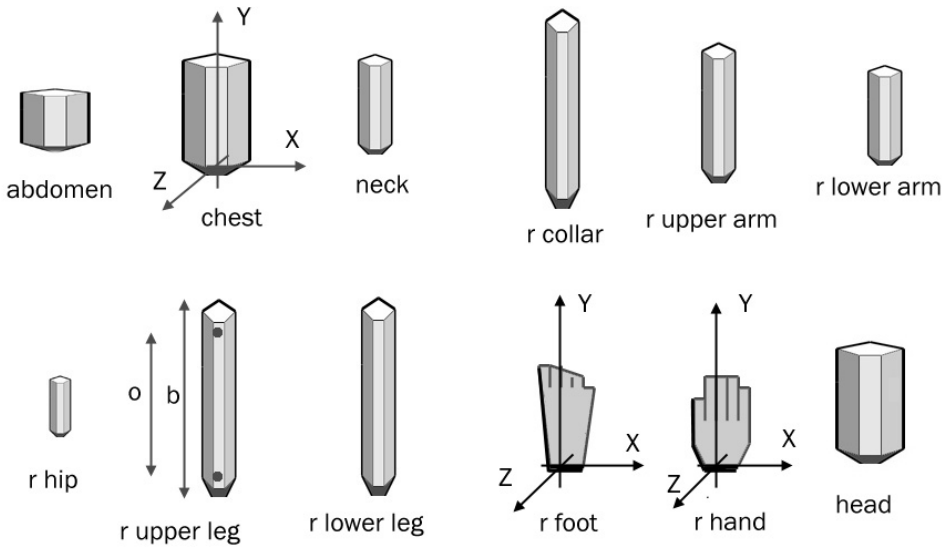


Figure 3.1: Collection of Disjointed `Bones`

Note: It is important that the figure is seen as a series of connections between the nodes. The bones themselves, in the .htr format for instance, can be offset from the node alignment and the bone length may be different from the distance between nodes. This reflects the fact that the sensors on the original motion capture subject will be on the outside of the body rather than at the internal joint; and that the bone may overrun its node. For instance if the foot is considered as a single bone, the heel

extends beyond and below the ankle joint. These offset details can be considered separately from the hierarchical skeleton construction, but add considerably to the liveliness of the motion and the rendered, clothed finished figure.

3.2. Constructing the figure in its neutral position

The bones are connected in a hierarchical form starting with a "global" parent which is placed at the origin of the global axes. A diagram of the hierarchical figure is shown earlier in section 1.3, figure 1.1.

The local axes of each bone are translated to a new origin, using the x, y, and z coordinates of its parent's offset values. The bone is again aligned along the local Y-axis. A figure will then be constructed in its neutral pose. As long as the bone length matches the Y-translation, the bones will appear to be connected. At the shoulders and hips an X-translation and Z-translation are also present, which shifts the bones sideways and disconnects them from their parent. This gap is filled with extra parts when the figure is rendered. No rotations are involved at this stage, so all the local axes remain parallel to the global axes.

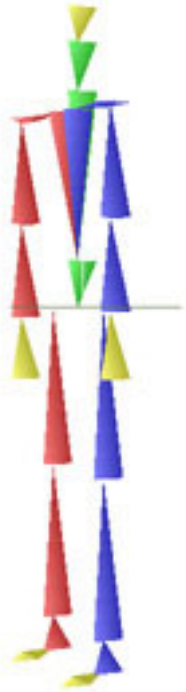


Figure 3.2: BVH Neutral Pose

This neutral pose data set provides a base position from which all subsequent poses can be generated as a set of frame data which is imposed frame-by-frame onto the

base data. In principle all that is needed for each frame is a translation of the global axes to a new position in 3D space, and a set of rotations at each node to shape the figure in the same way as an artist's mannequin is shaped.

This analogy with the wooden mannequin is important; just as the mannequin`s pose can only change by rotating the body parts at their joints, so the computer model rotates its bones pivoted from its parent`s node. This ensures that the body remains intact as it moves in a set of connected circular arcs with radii set by the bone lengths. Any attempt at local translations will tend to dismember the body. As has been said, in some systems, such as the .htr format, some translations subsequent to the base position are made, but these require carefully controlled dependencies between translations and rotations in order to maintain the integrity of the figure.

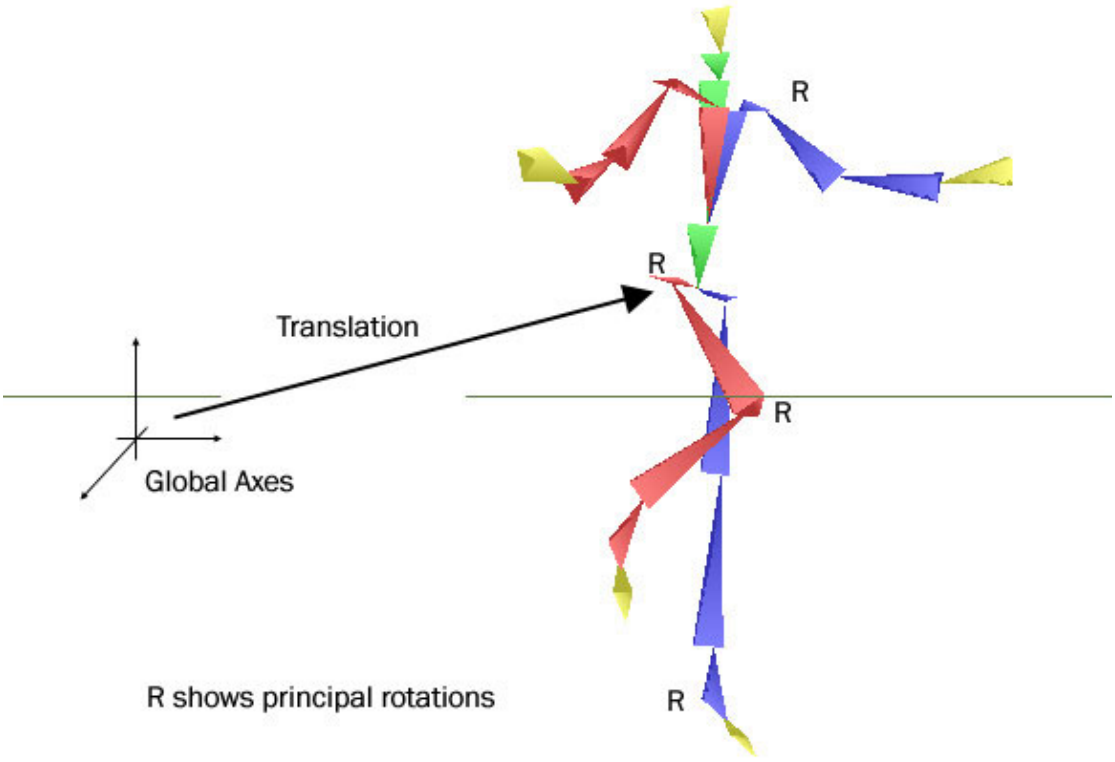


Figure 3.3: BVH Figure Translated and Rotated

3.3. Constructing figures using the .bvh and .htr formats

Note that this construction method uses a technique of transforming the local axes of each node and then placing its child`s bone on the Y-axis of the new axes. The hierarchical structure is therefore formed by starting at the global node and then working outwards towards the youngest children. This fits in with the OpenGL

graphics application which uses this method. Translations and rotations are expressed as matrices which operate on the current matrix. The current matrix defines the position of the current axes in relation to the world axes. The current matrix can be pushed onto a matrix stack and popped from the stack at a future time.

It is also possible to construct the figure by transforming the vertex of each node to a new position in the world coordinate system – the process is an exact reverse of the axes transformation as it starts from the youngest children and works inwards towards its parents.

3.3.1. The BioVision .bvh format

This is the most commonly supported motion capture file format, and a wide range of existing sequences is available. There is a large amount of flexibility in hierarchical forms, translational and rotational data, but this project has used a specific hierarchical format – the Character Studio Biped – which uses a specific base data set, and therefore limits the need for parsing all possible versions of the format.

The base data set for constructing the neutral pose contains 18 translation offsets which are used to position the nodes. The bone lengths are not specified, and would be presumed to connect between the nodes during the rendering process. The five youngest children (head, hands and feet) are drawn as extensions along their parental axes. The frame data includes an x, y, z translation vector to move the hips axes to the required position in the 3D world for each frame, and three rotation angles for each node, for each frame.

The figure is constructed as follows, using a stack of transformation matrices:

```
Push global matrix to be recalled for next frame
        Translate hips axes
        Rotate hips axes through y then x then z.
Push hip matrix
        Draw hip bone
        Translate axes to end of hip bone
        Rotate local axes through y then x then z.
Push chest matrix to be recalled for left collarbone
        Draw chest
        Translate axes to end of chest
        Rotate local axes through y then x then z.
        Draw neck bone
        Translate axes to end of neck bone
        Rotate local axes through y then x then z.
        Draw head
Pop chest matrix
Push chest matrix to be recalled for right collarbone
```

```
            Draw left collar bone
            Translate axes to end of left collar bone
            Rotate local axes through y then x then z.
            Draw left upper arm
            Translate axes to left elbow
            Rotate local axes through y then x then z.
            Draw left lower arm
            Translate axes to left wrist
            Draw left Hand
    Pop chest matrix
            Construct right arm as left
    Pop hips matrix
    Push hips matrix to be recalled for right leg
            Construct left leg as left arm
    Pop hips matrix
            Construct right leg as left leg
    Pop global matrix for next frame
```

The bones are drawn from the origin of the current axes (in practice the axes will have been rotated from the end of the parent`s bone) to the position defined by the bone`s base offset.

The .bvh system is very simple and safe, with no translations beyond the neutral pose apart from the global translation. It can provide more complex motion details as with the .htr format, but this simple version (the Character Studio Biped form) provides a good starting point for analysing motion capture.

3.3.2. The Motion Analysis .htr format

This format builds on the simple .bvh system and provides the following data for each node:

For the base data, x, y, and z translations, three rotation angles and the bone length.

For the frame data, x, y, and z translations, three rotation angles and a scale factor.

The rotation angles and bone length in the base data allows the construction of a neutral pose with bones placed in a more realistic way than for the .bvh format.

The construction process is similar to the .bvh method with some important exceptions; the translations are a sum of the base and frame translations for each node, and the rotations are performed in the order fx, fy, fz, bx, by, and bz (where bx represents the base rotation about the X-axis, fy the frame rotation about the Y-axis etc.). The bone is drawn according to its length, along the local Y-axis. As stated

above, this means that it may have a different alignment and length compared to the node (i.e. joint) positions.

3.3.3. Base Data for a Typical .htr File

```
[BasePosition]
#SegmentName Tx,     Ty,       Tz,       Rx,         Ry,         Rz,         BoneLength
Head         0.000003 141.9662  0.000002 -37.745777  -8.179454   6.203664    80.000046
Neck         0.000000 379.5667  0.000000  32.855431  -5.194619   1.823337    141.966263
Chest        0.000000 94.89169  0.000000  0.000002   12.877405   -0.000012   379.566803
LeftShould   0.000003 154.0236  0.000005  171.957962 -79.97757   -157.09491  215.219986
RightShold   -0.00000 146.4087  -0.000001 20.020029  60.865143   -8.207462   240.139038
LeftElbow    0.000007 215.2199  -0.000011 -24.162848 54.442085   -17.491318  336.440125
RightElbow   -0.00002 240.1390  0.000007  -21.925606 -43.61770   23.102211   322.101532
LeftWrist    -0.00001 336.4400  0.000002  173.178864 -0.531422   156.628021  100.348160
RightWrist   0.000005 322.1015  0.000026  1.868945   1.747480    14.125610   109.739998
Hips         377.8865 1077.530  704.4749  -179.01116 36.455761   -178.00671  94.891701
LeftHip      95.33676 5.238981  -8.33612  -0.485735  -9.415434   -176.11706  525.916443
RightHip     -96.1190 -4.69187  -4.85731  2.457280   -1.423929   -179.12556  528.302429
LeftKnee     0.000004 525.9164  0.000001  -10.646533 -14.54137   -1.855900   514.711975
RightKnee    -0.00002 528.3024  0.000004  -12.768950 2.118825    -3.063609   521.764099
LeftAnkle    0.000003 514.7119  0.000009  97.965164  -0.000001   0.000000    253.446518
RightAnkle   0.000001 521.7640  -0.00000  99.242271  0.000002    0.000000    256.283173
LeftCollar   19.99999 366.4832  -15.1268  169.660507 -4.312743   90.901894   154.023682
RightCollar-19.9999 366.4832  -15.1268  171.369568 4.171492    -91.645760  146.408707
```

Tx, Ty, and Tz provide the translational data for each segment, and Rx, Ry and Rz the rotational data. We will construct the figure using the method described for the .bvh figure, where the local axes are transformed and the bone then drawn at the Y-axis.

As in the diagram below, figure 3.4, the construction of the .htr neutral pose requires:

a. Translation of the hips axes from the global axes by TxTyTz.
b. Rotation of the hips local axes through Rx, Ry and Rz.
c. Translation of the right hip axes from the hips axes by TxTyTz.
d. Rotation of the right hip axes through Rx, Ry, Rz.
e. Repeat the process to set all the local axes positions.
f. Draw each bone aligned with its own local Y-axis.

As can be seen from the diagrams in figures 3.2 and 3.4, the main difference in the neutral poses is that the .htr system rotates the local axes. This means that the pose can be more realistic than the .bvh neutral pose, which can only provide small translations to an unrealistic `standing to attention` position. Having a realistic neutral pose is useful when rendering and clothing the figure.

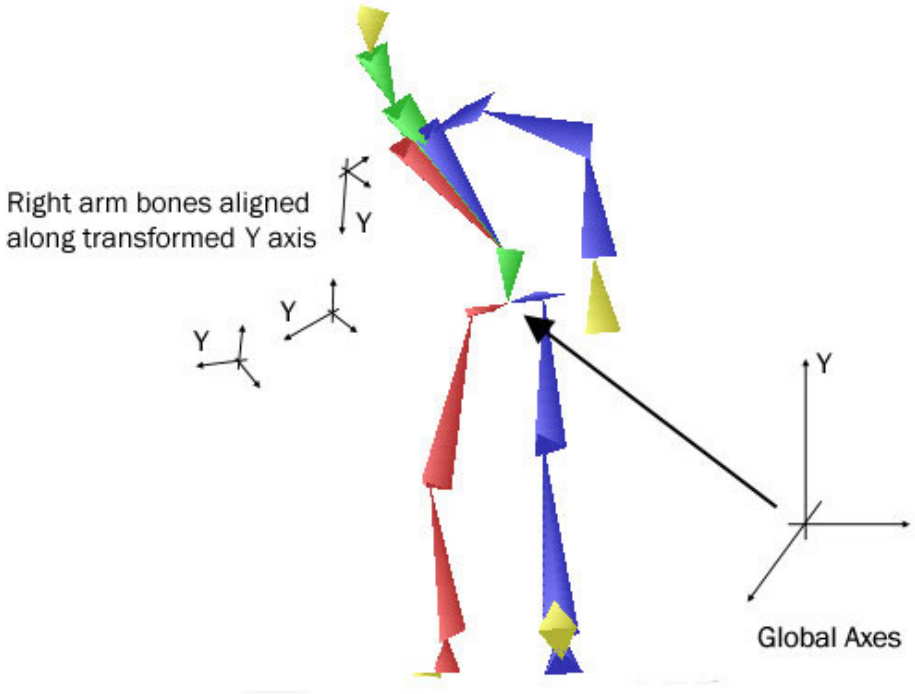


Figure 3.4: Constructing the HTR Neutral Pose

3.4. Performing Operations on the Motion Data

As long as the motion data is used in its original, unmodified form the resulting motion will be as expected. However, when the data is required to be modified or mixed with another sequence, a number of difficulties emerge, the main ones being:

1. Irregularities in the resulting rotation angles
2. Failure of the figure to relate correctly to the global axes
3. Attempts to mix incompatible motions

These and other problems must be investigated before successful modification and mixing can be done.

3.4.1. Irregularities in the Resulting Motion

Take the example of a clock face with an hour hand and a minute hand. The animator wants to add a third hand (drawn more boldly in the diagram) which is always half way between the other two hands. Starting at 12.00, the minute hand moves to half past, by which time the third hand has moved to quarter past. But as the minute hand moves on, what does the third hand do?

It has three sensible choices:

1. If the angle between the hour and minute hand is expressed as between +180° and –180°, and the third hand takes half this angle, then the third hand will move from the 12 to the 3, flip over to the 9 and move on to the 12.
2. If the angle between the hour and minute hand is expressed as 0° to 360°, halving this angle will move the third hand from the 12 to the 6, whence it will flip up to the 12 again.
3. If the angle between the hour and minute hand is expressed as 0° to 720° (i.e. a two hour period), the third hand will move smoothly round the clock.

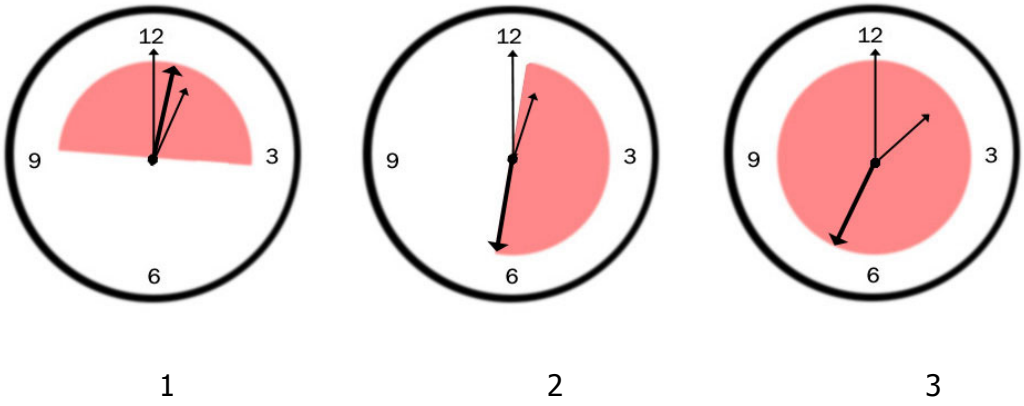


Figure 3.5: Example of Clock Motion

Which of these three approaches is correct? Only the animator can decide which method is more appropriate for the required effect, and it may be that the system cannot provide the information required to provide the two hour cycle. The shaded area shows the envelope of motion of the third hand.

Interpolating angles in motion capture confronts similar problems, especially when the angle moves through the 180° line as in the clock example. In the goal motions which were used by Jones and Dann, the figure flipped through 180° at different times in the motion. Repeating these mixes (using **`HTRMixer.exe`**) the same effect was seen.

Looking at the frame data for the hips, the Z rotation Rz moves through 180° between frames 46 and 47.

```
FR  Tx          Ty          Tz           Rx          Ry          Rz          SF
40  793.343018 -7.412964    -3232.192383 -170.614685  -81.995667 -163.23695 0.910511
41  792.329834 -25.116943   -3229.826416 -168.974625  -81.763672 -165.32449 0.908618
42  791.414185 -42.175476   -3227.177246 -167.131561  -81.757912 -167.78779 0.906564
43  790.703857 -58.527466   -3224.650635 -165.489044  -81.673164 -170.14918 0.905655
44  789.553833 -74.198608   -3221.209229 -163.844238  -81.755051 -172.52441 0.901626
45  788.723999 -88.567200   -3218.335449 -161.170792  -81.941429 -176.26855 0.899971
46  787.935669 -101.883057  -3215.869629 -158.413651  -81.639030 -179.26281 0.895039
47  786.029663 -114.294250  -3211.969971 -156.200638  -81.141022 178.042252 0.890678
```

```
48  784.500122 -125.460999  -3209.444092 -151.050293  -81.234192 172.574738 0.891316
49  782.311890 -134.867554  -3205.147949 -150.767487  -80.104622 172.185654 0.892165
50  780.591064 -143.377869  -3201.369629 -147.582230  -79.800026 169.015579 0.887781
51  778.967163 -149.926819  -3196.985840 -146.249954  -79.332130 167.584869 0.885188
52  776.954590 -154.739746  -3191.703613 -145.888077  -78.475609 167.151245 0.880680
```

In this situation the irregularity was removed by adding a condition which added 360° to the angle if it was less than zero:

```
float check180(int nd, float angle)
{
   if(angle<-0 & nd==8) angle = 360 + angle;
   return angle;
}
```

but this was found to be appropriate only at the hip (node 8). At other nodes, changing the angle in this way produced unwanted motions. This confirms what is suggested from the clock example, that the animator needs to make judgements about the appropriateness of these constraints in order to give good results.

Problems in interpolating angles are compounded when two or three rotations are multiplied together. The combination of three fixed axis angles are called **Euler Angles** after Leonhard Euler (1707-1783). In his work *Theoria motus corporum solidorum* he decomposed the motion of a solid into a rectilinear motion and a rotational motion (i.e. translation and rotation), as part of his investigations in cartography and the motion of celestial bodies.

3.4.2. Interpolating Euler Angles

The objective of the angle rotations is to present each element of a figure (a bone in this context) at a particular orientation in relation to its axes. In the diagram we start with the element aligned on its Y-axis, facing positively down the Z-axis. The element is rotated firstly by -60° about the X-axis, then by 30° about the new Y-axis, and then by 40° about the new Z-axis (Euler A).

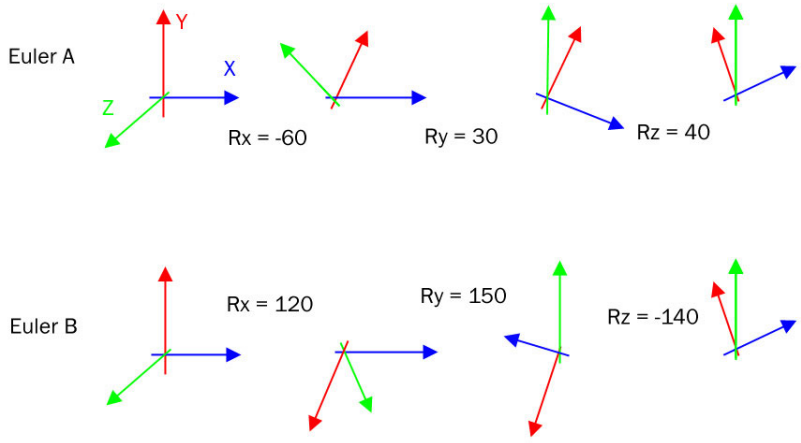


Figure 3.6: Arriving at Similar Orientations with Different Euler Angles

Significantly, the resulting orientation can also be achieved by other combinations of Euler angles. For instance, changing the angles from (rx. ry, rz) to (rx+180, 180-ry, rz-180) will give the same final orientation, unnoticed by the viewer of the final motion sequence (Euler B). This has been tested using the program **`EulerTest180.java`**. Although it is possible to change randomly between these two representations during a motion sequence with no noticeable effects, this will not be true if the angles are interpolated.

Consider the above example using the Euler angles (-60, 30, 40). Dividing each angle by half (compare the Clock example) gives (-30, 15, 20) which will produce an intuitively correct orientation which is in the same approximate direction, but closer to the original axes. However, if the angles are presented as (-60+180, 180-30, 40-180), halving will give Euler Angles of (60, 75, -140) which will not give the expected orientation.

Although it would be appear to be unlikely that these changes in representation will occur, they actually happen frequently due to the processes used to interpret the original motion capture data. An example is shown here where the Euler angles for the left knee change between frames 58 and 59.

```
WalkLoop89.bvh leftknee

                Ry             Rx             Rz
Frame: 50      43: 3.72       44: 46.08      45: -5.28
Frame: 51      43: 3.49       44: 47.52      45: -4.81
Frame: 52      43: 4.45       44: 48.85      45: -5.22
Frame: 53      43: 5.56       44: 50.44      45: -5.17
Frame: 54      43: 5.83       44: 53.06      45: -3.89
Frame: 55      43: 5.70       44: 57.37      45: -2.52
Frame: 56      43: 6.01       44: 63.73      45: -2.01
Frame: 57      43: 6.78       44: 72.20      45: -2.14
Frame: 58      43: 6.75       44: 82.53      45: -1.32
Frame: 59      43: -164.05    44: 86.21      45: 170.61
Frame: 60      43: -166.92    44: 75.99      45: 174.01
```

This change has no effect on the motion, as can be seen from the following data which shows the XYZ vertex for the Left Foot, using the program **`BVHPrint.java`**.

```
C:\myjava\mocap>java BVHPrint WalkLoop.bvh
89
WalkLoop.bvh: LeftFoot(x,y,z)
50:            0.18582375   5.364949      -2.7543812:
51:           -0.3788225    5.5025644     -2.2120905:
52:           -1.0170637    5.7467465     -1.4560955:
53:           -1.9440917    6.17472       -0.6067536:
54:           -3.1617177    6.872692      0.42915076:
55:           -4.452238     7.9506893     1.7080128:
56:           -5.6059985    9.546896      3.211148:
57:           -6.4593887    11.709507     4.919388:
58:           -6.872312     14.217673     6.9465127:
59:           -6.8579736    16.49652      9.390071:
60:           -6.4041123    17.881552     12.064581:
```

It can be seen that the left foot vertex moves from (-6.872312, 14.217673, 6.9465127) to (-6.8579736, 16.49652, 9.390071) between frames 58 and 59, which is an acceptably smooth motion. However, when the angles are divided by two, using the program **`D3.exe`**, the motion shows an idiosyncratic leg kick in the left leg caused by the difference in angle representation, as shown in figure 3.7.

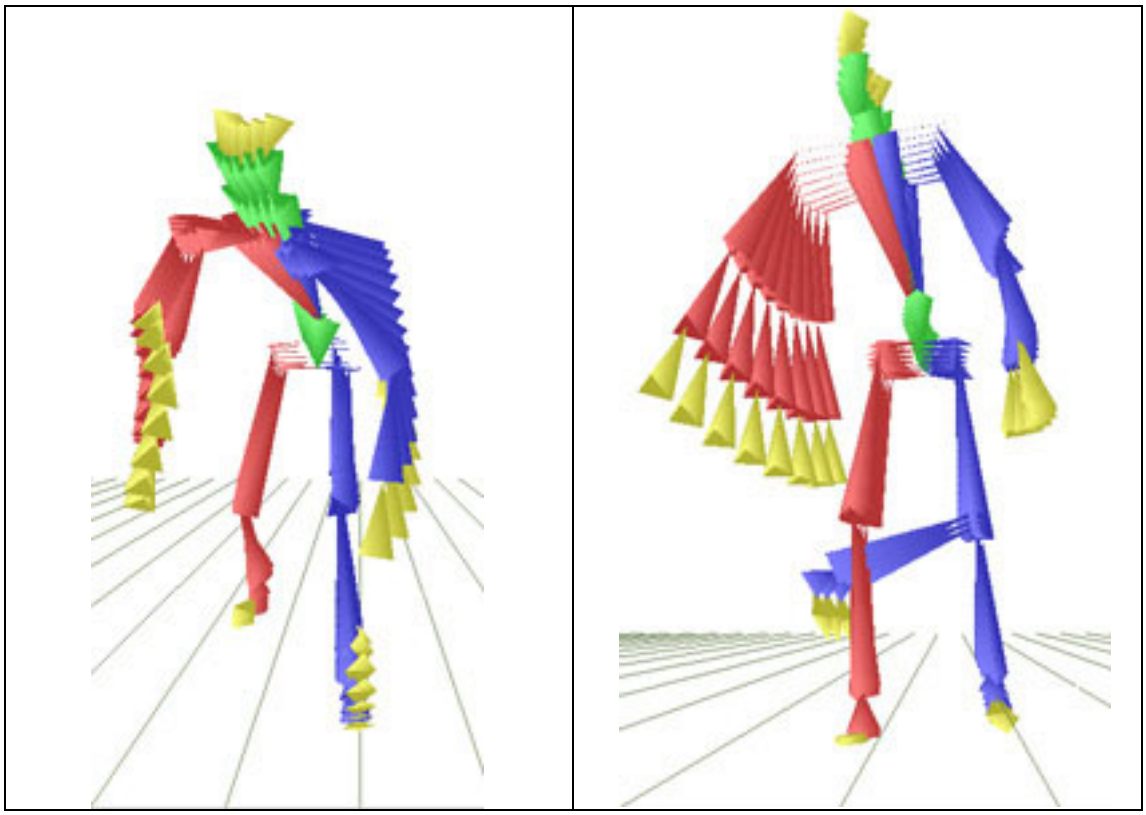


Figure 3.7: Idiosyncratic Interpolated Motion

Rectifying these problems is more difficult than a simple condition such as the "360-angle" option. A more detailed analysis of angle representation is required, and will be dealt with in the next chapter.

3.4.3. Failures of the Figure to Relate Correctly to the Global Axes

The most obvious and persistent example of this is that the foot often fails to make proper contact with the floor in interpolated motions. The figure then appears to be walking on a springy mattress rather than on a solid floor. This can be seen in the following figure, where a sideways view of the modified walk shows the feet pushing through the floor. This contrasts with the original motion where the foot is stationary once it hits the floor.

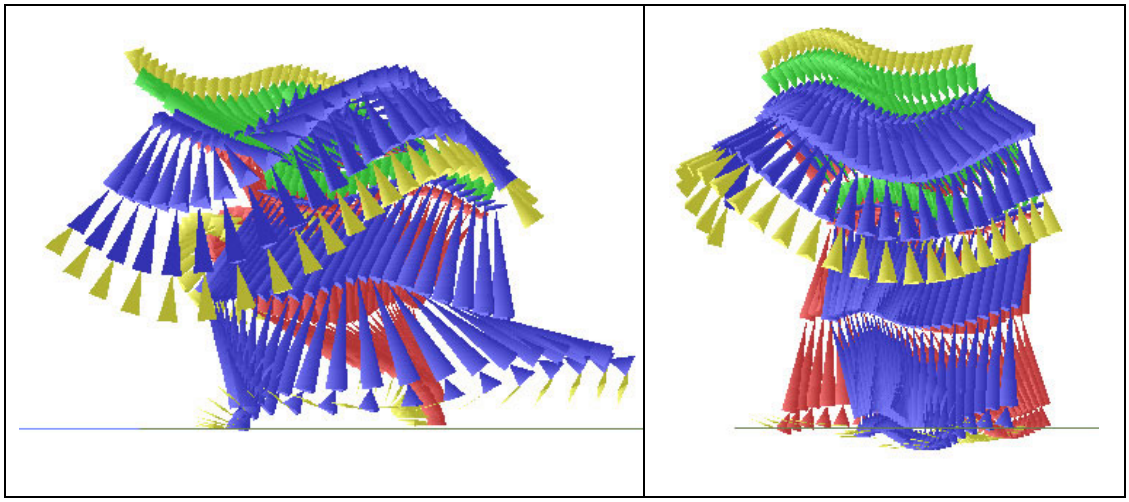


Figure 3.8: Problems with the Feet not Hitting the Floor

This is a general problem when interpolating rotations, as can be seen from the following simple example where a single jointed figure is standing upright on the floor, with its global node at the top. In the hierarchical system this figure is constructed with a Y-translation from the global axes of 0, a Z-rotation of 180°, a Y-translation of 1, a Z-rotation of 0° and a Y-translation of 1, at which point the bottom of the figure is placed on the floor.

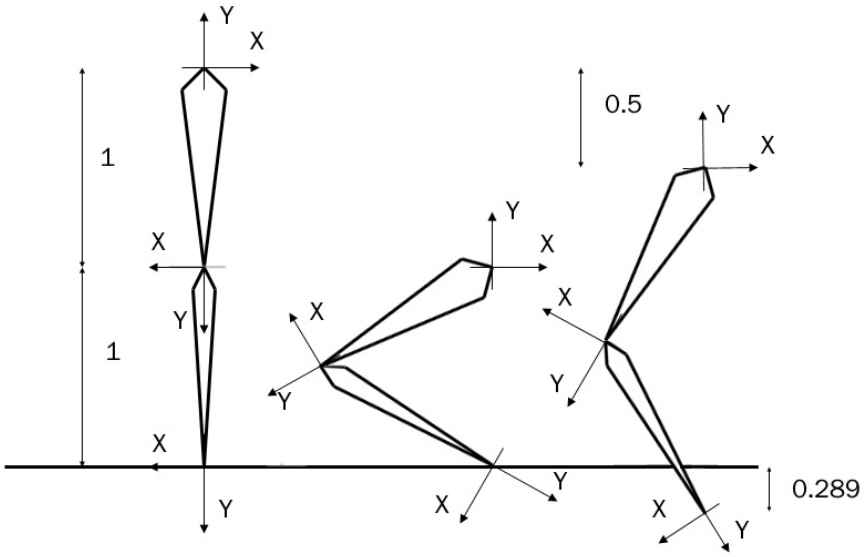


Figure 3.9: Simple Figure Interpolation

A second frame has the figure bending at the "knee" by transforming with a Y-translation from the global axes of 1, a Z-rotation of -60°, a Y-translation of 1, a Z-rotation of -60° and a Y-translation of 1. The figure again stands on the floor.

A 50 per cent interpolation of these two figures (with a Y-translation from the global axes of 0.5, a Z-rotation of -30°, a Y-translation of 1, a Z-rotation of -30° and a Y-translation of 1) results in a pose which does not rest on the floor, as can be seen in figure 3.9.

Correcting these problems is not simply a matter of moving the global translation by a suitable amount so that the foot is effectively clamped to the floor in the required frames. This would result in a jerky motion where the figure suddenly shifts vertically when it is released from the floor. The only solution is to use **Inverse Kinematics**, which is outside the scope of this project, and defeats the object of using interpolations to mix motion sequences.

Within this project the only solution is to carefully choose the mocaps so that these faults either do not occur or at least do not offend the viewer.

3.4.4. Attempts to Mix Incompatible Motions

A more obvious source of problems is when the two motions do not match in the time period of their motion. A trivial example is where a running loop is mixed with a walking loop resulting in a comical "silly walk". Even when the match is closer, careful attention has to be given to the synchronisation of limb movements, and sliding effects due to different translations of the global node.

It is particularly difficult to match actions where the figure jumps off the floor. Here the laws of gravity apply and any changes to the path or period of the motion will be noticed as being infeasible.

In practice this means that the two motions need to have the same time period and phase, and the actions below the hips need to be very similar. More differences can be accommodated in the motion of the arms and upper body.

3.5. Representation of Rotation Angles

The data required to transform the figure through its different orientations can be represented in a number of forms, and four are discussed here; Euler angles, homogeneous matrices, axis/angle pairs, and quaternions. The requirement common to these data types is that each element of the figure can take up any orientation in three-dimensional space.

3.5.1. Euler Angles

In many ways Euler angles are the natural way to express these rotations; they are easy to visualise, use a sequence of only three numbers, and allow freedom constraints to be easily applied. When motion sequences are produced the rotation of each joint is constrained in order to mimic real body actions. For instance, the knee joint should only rotate in the X-axis by about 120°, and only very small rotations in the other axes should be allowed. As Euler angles are stored as x, y, and z rotations these constraints can be easily applied.

The disadvantage of Euler angles is that interpolation causes difficulties. As has been shown above, linear interpolations can give unwanted results as the three rotations are operated on independently.

Because there is more than one way to move between two orientations using Euler angles, a decision has to be made as to which one is most appropriate. Sometimes it is impossible to arrive at the desired orientation due to the well-known problem of Gimbal Lock, named after the gimbal on an aircraft which should maintain a horizontal orientation when the plane turns and spins in the air. Unfortunately some combinations of moves lock the gimbal so that it cannot right itself horizontally, causing the pilot to be disorientated. For instance if the second rotation is by 90°, the third rotation is actually performed about the axis of the first rotation. This is a

problem when proceeding from one Euler angle to the next in sequence, but does not mean that some orientations are impossible to store as Euler angles.

In practice the advantage of Euler angles as the storage data type outweigh the disadvantages, which can be remedied by converting to other more suitable data types for processing.

3.5.2. Homogeneous Matrices

4x4 homogeneous matrices are multi-purpose operators which provide rotation, translation, scaling and shearing. As such they are extensively used in graphics applications (such as OpenGL and Java3D) as the main data type in which operations are actually performed.

However, because they are multi-purpose they need careful control to ensure that they only perform the required operations; scaling or translation effects should not be present if only rotations are required.

Matrices are also unwieldy and wasteful data types. Three rotations require between 9 and 48 numbers depending on whether the three rotations are combined into one matrix and whether constant numbers are stored. Multiplying rotation matrices can be speeded up by operating on the pair of coordinates for each axis in turn, as used in the transform() method used throughout this project:

```
oldX=x; oldY=y; oldZ=z;
x = oldX*Math.cos(ry) + oldZ*Math.sin(ry);
z = -oldX*Math.sin(ry) + oldZ*Math.cos(ry);
y = oldY;
oldX=x; oldY=y; oldZ=z;
y = oldY*Math.cos(rx) - oldZ*Math.sin(rx);
z = oldY*Math.sin(rx) + oldZ*Math.cos(rx);
x = oldX;
oldX=x; oldY=y; oldZ=z;
x = oldX*Math.cos(rz) - oldY*Math.sin(rz);
y = oldX*Math.sin(rz) + oldY*Math.cos(rz);
z = oldZ;
x=x+tx;
y=y+ty;
z=z+tz;
```

The process can be speeded up further by precalculating the cos and sin values.

3.5.3. Axis/angle pairs

Like Euler angles, axis/angle pairs give an intuitive visualisation of rotating figures. Firstly an **axis** is constructed between the origin and a vertex in Euclidean space

represented as (x, y, z). The figure spins about this axis through the **angle** θ specified in the axis/angle pair. The pair is written as (θ, (x, y, z)).

The axis/angle pair is used in OpenGL in the function **`glRotate*(`*`angle`*`, x, y, z),`** which passes an axis/angle pair.

Problems occur in the linear interpolations of axis/angle pairs. There could be an unfortunate situation where the interpolated vertex is at the origin, giving an impossible axis between two vertices both at (0, 0, 0). For this reason, and to improve smoothness, the interpolations should follow a curved line on the surface of a notional sphere, as explained below in reference to quaternions..

Parameterization using axis/angle pairs is not as successful as using quaternions, which perform interpolations and matrix conversions quicker, although the two types are closely related, and many operations can be used with either type.

3.5.4. Quaternions

A quaternion is written as (w, (x**i**+y**j**+z**k**)) where (x**i**+y**j**+z**k**) is a 3D vector and w is a scalar quantity. An axis/angle pair (θ, (x, y, z)) converts to a quaternion as

$$q = (\cos(\theta/2), (x*\sin(\theta/2), y*\sin(\theta/2), z*\sin(\theta/2))$$

and when $\theta=\pi$,

$$q = (0, (x, y, z)),$$

which is a pure quaternion equivalent to a 3D vector. Only unit quaternions express rotation, when the magnitude is 1 and is expressed similarly to vectors as

$$|q| = \sqrt{(w^2+x^2+y^2+z^2)}.$$

Quaternions have a ready-made algebra which was developed by their inventor, Sir William Hamilton, and they have proved to be the most useful way of expressing single-axis rotation. It is easy to convert from the stored Euler angles, interpolate the quaternions, and then convert the resulting quaternions to rotation matrices for display in a graphics application. This procedure is used in the software described in a later chapter.

Spherical Linear Interpolating or SLERPING allows smooth interpolation along the surface of the four dimensional quaternion sphere. A standard slerping algorithm is included in the software listings.

3.6. Conclusions on Rotation Representation

Euler angles are the most suitable type for file storage, being compact (only three sequential numbers, the fourth parameter of bone length being stored only once as a constant), and being suitable for constraining joint movements. Euler angles can also be used to interpolate motions as long as the data is consistent in its expression of orientation. Even then there are likely to be problems, and quaternions are more suitable for interpolation.

Finally, the rotations are converted to transformation matrices, as these are used for display in most current graphics applications.

Chapter 4:Designing and Building the Software

4.1. Design Requirements

The computer programs had the following requirements:

1. To parse .bvh and .htr mocaps taken from file.
2. To display the mocaps in a desktop window.
3. In the display, to clearly see the motion of the figure.
4. In the display, to clearly see the orientation of the bones.
5. To see the relationship between the figure and the global axes.
6. To change the viewing angle of the display.
7. To slow down the frame rate.
8. To print out vertex coordinates of specified nodes.
9. To mix two sequences by interpolating key parameters.
10. To offset the start points to compare mixed results.
11. To perform the interpolations using Euler angles and quaternions.
12. To select and if necessary modify two suitable goalkeeping mocaps.
13. To calculate and display the required envelopes.
14. To store the envelope data in a suitable data type.
15. To move the ball through 3D space.
16. To move the goalkeeper in a merged motion.
17. To calculate whether the save has been made.
18. To continue the motion with the ball in hand or moving past the goalkeeper.
19. To randomly repeat the action.

In addition, printline statements will be added to the programs when required to analyse the results.

4.2. Parsing and Displaying the Files

The first programs were written in Java using Swing. A class **`XYZ`** was created which holds data and methods for each node of the figure. The main features of this class is that its x, y, z coordinates are calculated recursively through its parents, grandparents etc., with the coordinates always expressed relative to the world axes.

This means that a trace of the motion can be plotted in 3D, and printouts taken of the absolute coordinates at any node (see Requirement 8). This algorithm was also used later to calculate whether the goalkeeper`s save has been made (Requirement 17).

The following recursive algorithm is performed by a call **`myXYZ.trans()`**:

```
public void trans() {
    x=0;y=0;z=0;              //reset vertex to global origin
    transform(this);      //start the recursion
}

private void transform(XYZ child) {

    float[] m = child.getMatrix();

    rotate(frx, fry, frz);             //rotate frame data
    rotate(brx, bry, brz);             //rotate base data
    x=x+btx+ftx;                              //translate x
    y=y+bty+fty;                              //translate y
    z=z+btz+ftz;                              //translate z

    if(child.getParentName() != GLOBAL)
      transform(child.getParent());
}

private void rotate(float rx, float ry, float rz) {

     oldX=x; oldY=y; oldZ=z;
     x = oldX*Math.cos(ry) + oldZ*Math.sin(ry);
     z = -oldX*Math.sin(ry) + oldZ*Math.cos(ry);
     y = oldY;
     oldX=x; oldY=y; oldZ=z;
     y = oldY*Math.cos(rx) - oldZ*Math.sin(rx);
     z = oldY*Math.sin(rx) + oldZ*Math.cos(rx);
     x = oldX;
     oldX=x; oldY=y; oldZ=z;
     x = oldX*Math.cos(rz) - oldY*Math.sin(rz);
     y = oldX*Math.sin(rz) + oldY*Math.cos(rz);
     z = oldZ;
     x=x+tx;
     y=y+ty;
     z=z+tz;
}
```

These programs (**`HTR.java`** and **`BVH.java`**) provided good means of analysing the data by plotting each vertex. However, Java Swing did not provide fast enough graphics routines to give a smooth animation (Requirement 3). Also, as the figures were drawn by connecting the vertices together, there was no means of seeing the orientation of the bones, i.e. which way the bone faced along its Y-axis (Requirement 4).

For these reasons, subsequent programs were written in C++ using OpenGL graphics.

OpenGL provides a library of functions which are used to draw objects in 3D space. Objects are drawn relative to the current axes in this space, and the axes can be moved by applying a transformation matrix to the current matrix. In contrast with the vertex operations described above, drawing the figure here involves starting at the global axes and continually drawing the bone and moving to the child`s axes. Transformations are performed in **exactly the opposite order** compared with the vertex operations used in **`HTR.java`** and **`BVH.java`**.

Because we are continually updating the current axes we lose the relationship with the global axes, which means that there is no vertex data available, which is needed for Requirements 8 and 10. To remedy this a vertex algorithm is also used where required.

The OpenGL functions were found to operate quickly enough to allow a simple rendering function to be incorporated which gives a good level of visualisation. This fulfills Requirements 3-7, displaying the figure, its motion and orientation clearly at different frame rates and different viewing angles.

The figure was rendered using the **GL_TRIANGLE_FAN** object, with the bone being drawn at the current axes which were then translated to the end of the bone:

```
glColor3f(r, g, b);
glBegin(GL_TRIANGLE_FAN);

          glVertex3f(0.0, 0.0, 0.0);
          glVertex3f(x-1.5, y, z-1.5);
          glVertex3f(x+1.5, y, z-1.5);

          r=r+0.3; g=g+0.3; b=b+0.3;   //lighten colour
          glColor3f(r, g, b);
          glVertex3f(x, y, z+1.5);
          glVertex3f(x-1.5, y, z-1.5);

glEnd();

}
glTranslatef(x, y, z);
```

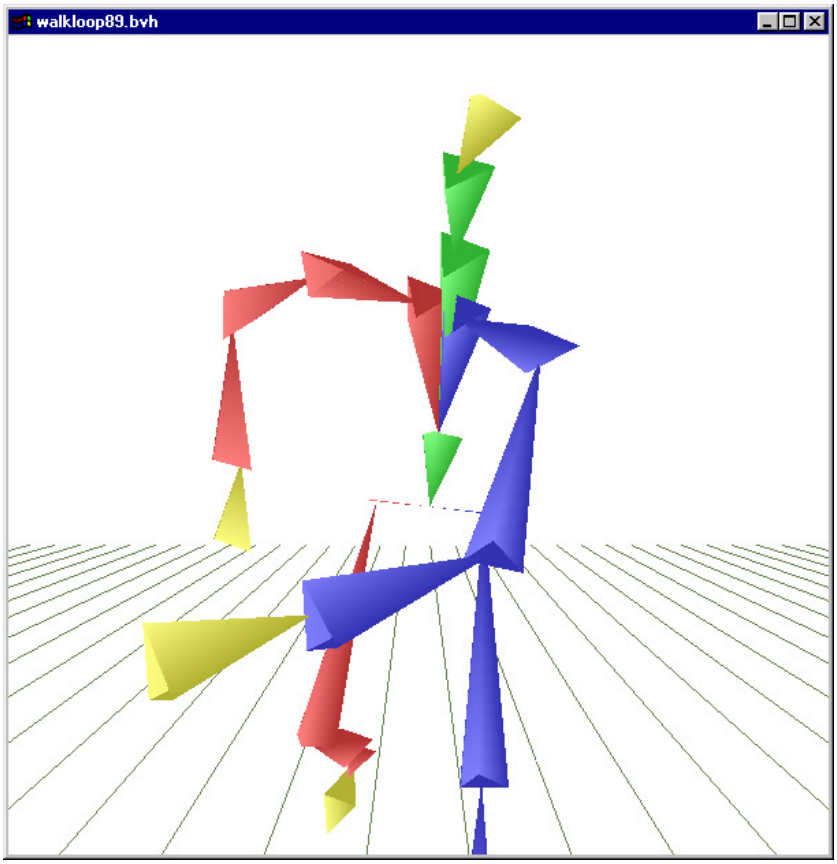


Figure 4.1: Rendered Figure

4.3. Mixing Sequences

The simplest form of mixing is the one used by Michael Jones (2000) and Stephen Dann (2001) where the frame data for sequences A and B is combined in the form:

dataA*i + dataB*(1-i);

where i is the interpolation factor, between 0 and 1.

The program **`HTRMixer.exe`** modifies the standard function **`rotateBone()`** to include interpolation:

```
void rotateBone(int nd, int fr)
{
	glRotatef(bd[nd][5], 0, 0, 1);
	glRotatef(bd[nd][4], 0, 1, 0);
	glRotatef(bd[nd][3], 1, 0, 0);

	glRotatef(md[fr][nd][5])*pc + md2[fr][nd][5]*(1-pc), 0, 0, 1);
	glRotatef(md[fr][nd][4])*pc + md2[fr][nd][4]*(1-pc), 0, 1, 0);
	glRotatef(md[fr][nd][3])*pc + md2[fr][nd][3]*(1-pc), 1, 0, 0);
}
```

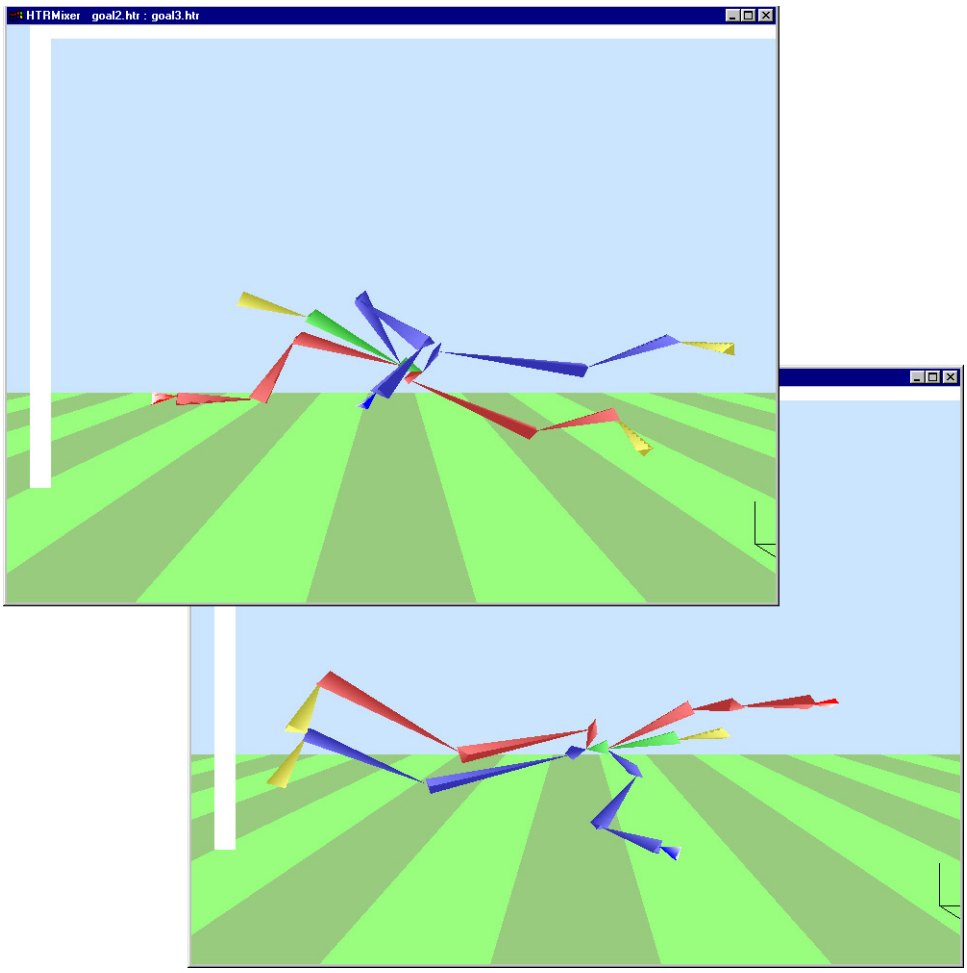


Figure 4.2: `Flipping` Effects

The display showed the problems of interpolation due to the angle data passing through 180°; the figure performs 180° flips about the hip axes (figure 4.2).

The program was then modified by the algorithm

```
if(angle<-0 & nd==8) angle = 360 + angle
```

which lessened the "flipping" problem but did not remove it entirely.

A further modification was made to allow a frame offset between the two sequences, meaning that the sequences were mixed at different points in the motion. This was to investigate whether the motions could be synchronised more accurately.

With these controls it was possible to find two mocaps, goal2.htr and goal3.htr which mixed correctly when offset by 20 frames. These were used in the final catching program.

However, other mixes did not work, and this solution is obviously limited to a few closely related motions. It should also be noted that the base data must be identical

in both sequences, and that the translation frame data should also be interpolated, otherwise the motion slides from one translation to another.

4.4. Interpolating with Quaternions

Program **`HTRQ.exe`** revises the function **`rotateBone()`** to take the rotations into and out of quaternion space. The sequence of operations is:

1. convert base Euler angles to quaternions
2. convert sequence A frame Euler angles to quaternions
3. convert sequence B frame Euler angles to quaternions
4. multiply base and A frame quaternions
5. multiply base and B frame quaternions
6. interpolate two resulting quaternions
7. convert interpolated quaternion to rotation matrix

The following functions were used, partly based on work by Alan Watt and Mark Watt (1992) and Nick Bobick (2001).

Euler angle (rx, ry, rz) to quaternion (q[0], (q[1], q[2], q[3])) conversion:

```
cx = cos(rx/2);
cy = cos(ry/2);
cz = cos(rz/2);
sx = sin(rx/2);
sy = sin(ry/2);
sz = sin(rz/2);

q[0] = cx * cy * cz + sx * sy * sz;
q[1] = sx * cy * cz - cx * sy * sz;
q[2] = cx * sy * cz + sx * cy * sz;
q[3] = cx * cy * sz - sx * sy * cz;
```

Quaternion multiplication qbqa:

```
A = (qa[0]+qa[1])*(qb[0]+qb[1]);
B = (qa[3]-qa[2])*(qb[2]-qb[3]);
C = (qa[0]-qa[1])*(qb[2]+qb[3]);
D = (qa[2]+qa[3])*(qb[0]-qb[1]);
E = (qa[1]+qa[3])*(qb[1]+qb[2]);
F = (qa[1]-qa[3])*(qb[1]-qb[2]);
G = (qa[0]+qa[2])*(qb[0]-qb[3]);
H = (qa[0]-qa[2])*(qb[0]+qb[3]);

q[0] = B+(-E-F+G+H)/2;
q[1] = A-( E+F+G+H)/2;
q[2] = C+( E-F+G-H)/2;
q[3] = D+( E-F-G+H)/2;
```

Quaternion (q[0], (q[1], q[2], q[3])) to matrix mat[16] conversion:

```
x2 = q[1] + q[1]; y2 = q[2] + q[2]; z2 = q[3] + q[3];
```

```
xx = q[1] * x2; xy = q[1] * y2; xz = q[1] * z2;
yy = q[2] * y2; yz = q[2] * z2; zz = q[3] * z2;
wx = q[0] * x2; wy = q[0] * y2; wz = q[0] * z2;
mat[3]=0;mat[7]=0;mat[11]=0;mat[12]=0;
mat[13]=0;mat[14]=0;mat[15]=1;
mat[0]=1.0 - (yy + zz);
mat[4]=xy - wz;
mat[8]=xz + wy;
mat[1]=xy + wz;
mat[5]=1.0 - (xx + zz);
mat[9]=yz - wx;
mat[2]=xz - wy;
mat[6]=yz + wx;
mat[10]=1.0 - (xx + yy);
```

Initially the program was written with no interpolation, meaning that the quaternion conversions should be transparent. **Testing with single .htr files showed displayed motion which was identical to that without the quaternion conversions, suggesting a successful process. However, once linear interpolation was included the "flipping" effects seen in Euler angle interpolations were still present.**

A simple program **`QuatTest.java`** was written to examine the Euler to quaternion conversions, and three Euler angle triplets – (0, 180, 180), (180, 0, 0), (-180, 0, 0) – were converted:

```
C:\myjava\mocap>java QuatTest 0 180 180
qw = 0.0
qx = -1.0
qy = 6.123233995736766E-17
qz = 6.123233995736766E-17
magnitude = 1.0

C:\myjava\mocap>java QuatTest 180 0 0
qw = 0.0
qx = 1.0
qy = 0.0
qz = 0.0
magnitude = 1.0

C:\myjava\mocap>java QuatTest -180 0 0
qw = 0.0
qx = -1.0
qy = 0.0
qz = 0.0
magnitude = 1.0
```

This shows that an X rotation of 180° gives qx = 1 whereas an X rotation of -180° gives qx as –1, hence the "flipping" effect.

More research is required to check these results and find suitable remedies.

4.5. Producing the Mixed Catch Sequences

Having selected and modified two suitable catching sequences, as required in Requirement 12, the next step (Requirements 13 and 14) is to create envelopes and store them in a suitable data form.

4.5.1. Creating the Envelopes

It was decided to use the right hand position as the basis of the envelope, and a right hand vertex was created for each frame of the given sequence. This was achieved by using the vertex transformation algorithm which calculates the vertex in relation to the global axes. This is important if the results are to be compared to other positioned objects, such as the ball, in the global space.

Three arrays **`envX[f][mix]`**, **`envY[f][mix]`**, **`envZ[f][mix]`** were created to hold the right-hand vertex for each frame for each of the mix proportions. Eleven mixes 0, 1, 2….10 were used here. The vertices were calculated by transformations as in the programs **`BVH.java`** and **`HTR.java`**.

```
pc=1;
for(mix=0;mix<11;mix++) {
       pc=pc-0.1;
       for(f=fo;f<frames;f++) {
              xp=0.00f; yp=0.00f; zp=0.00f;

              //update xp, yp and zp
              transformPoint(7, f);
              transformPoint(5, f);
              transformPoint(3, f);
              transformPoint(1, f);
              transformPoint(8, f);

              envX[f][mix] = xp;
              envY[f][mix] = yp;
              envZ[f][mix] = zp;
       }
}
```

The envelope-drawing routine retrieves **`envX[][]`**, **`envY[][]`**, and **`envZ[][]`** to display the envelope:

```
glPointSize(5);                                  //draw envelope
glBegin(GL_POINTS);
for(mix=0;mix<11;mix++) {
       glColor3f((float)(10-mix)/10, 0.70f, (float)(mix)/10);
       for(f=fo;f<frames;f++) {
              xp = envX[f][mix];
              yp = envY[f][mix];
              zp = envZ[f][mix];
              glVertex3f(xp, yp, zp);
       }
}
glEnd();
```

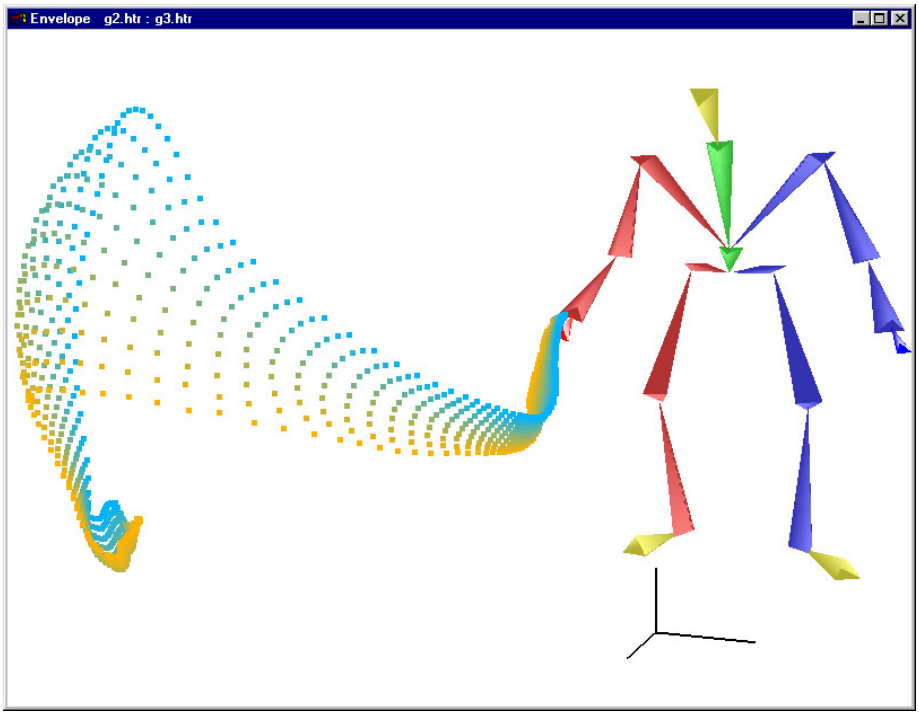


Figure 4.3: The Mocap Envelope

4.5.2. Catching the Ball

For the program **`Keeper.exe`** a **`caught`** boolean was used which was set if the right hand vertex **`(xp, yp, zp)`** was close to the ball vertex (**`ballX, ballY, ballZ`**):

```
if(    (xp > ballX-1) &
       (xp < ballX+1) &
       (yp > ballY-1) &
       (yp < ballY+1) &
       (zp > ballZ-1) &
       (zp < ballZ+1) )

       caught = true;
```

4.5.3. Moving the Ball

In order to provide a workable interactive catch program it was decided to place the ball at one of the envelope points. The ball is then moved to that point and if the **`caught`** boolean is set the ball follows the vertex of the figure`s right hand, giving the impression that the ball is being held. Otherwise the ball continues on its trajectory behind the figure.

The following functions are used to provide these actions:

```
void drawFootball() {
       float xball = ballX-((f-shootFrame)*10);
       float yball = ballY;
       float zball = ballZ;
       icos(xball, yball, zball);
}


int random(int rnd) {
       return rand()*rnd/RAND_MAX;
}


void shoot() {
       shootFrame = 70 + random(40);
       int mixRand = random(11);
       ballX=envX[shootFrame][mixRand];
       ballY=envY[shootFrame][mixRand];
       ballZ=envZ[shootFrame][mixRand];
}


void carryFootball() {
       icos(-2.00f, 3.00f, 0.00f);
}
```

In addition an **`autoSave()`** function is provided in which the figure automatically catches the ball:

```
void autoSave() {
       int fs=0;
       mix=0;
       while(
                !((envX[fs][mix] > ballX-1) &
                  (envX[fs][mix] < ballX+1) &
                     (envY[fs][mix] > ballY-1) &
                  (envY[fs][mix] < ballY+1) &
                  (envZ[fs][mix] > ballZ-1) &
                  (envZ[fs][mix] < ballZ+1)) &
                mix<10) {
              fs++;
              if(fs>(frames-1)) {
                     fs=fo;
                     mix++;
              }
       }
       pc=(float)mix/10;
}
```

The ball is drawn as an icosahedron using a routine adapted from Mason Woo, Jackie Neider and Tom Davis (1997):

```
void icos(float ix, float iy, float iz) {
  float xi = .52573f;
  float zi = .85065f;

  float vtx[12][3] = {
    {-xi, 0.0, zi}, {xi, 0.0, zi}, {-xi, 0.0, -zi}, {xi, 0.0, -zi},
    {0.0, zi, xi}, {0.0, zi, -xi}, {0.0, -zi, xi}, {0.0, -zi, -xi},
    {zi, xi, 0.0}, {-zi, xi, 0.0}, {zi, -xi, 0.0}, {-zi, -xi, 0.0}
    };
```

```
  int vty[20][3] = {
    {0,4,1}, {0,9,4}, {9,5,4}, {4,5,8}, {4,8,1},
    {8,10,1}, {8,3,10}, {5,3,8}, {5,2,3}, {2,7,3},
    {7,10,3}, {7,6,10}, {7,11,6}, {11,0,6}, {0,1,6},
    {6,1,10}, {9,0,11}, {9,11,2}, {9,2,5}, {7,2,11}
    };

  glBegin(GL_TRIANGLES);

  for(int ic=0; ic<20; ic++) {
    glColor3f(1.00f, 1.00f, (float)(20-ic)/20);
    glVertex3f(ix+vtx[vty[ic][0]][0],
               iy+vtx[vty[ic][0]][1],
               iz+vtx[vty[ic][0]][2]);
    glVertex3f(ix+vtx[vty[ic][1]][0],
               iy+vtx[vty[ic][1]][1],
               iz+vtx[vty[ic][1]][2]);
    glVertex3f(ix+vtx[vty[ic][2]][0],
               iy+vtx[vty[ic][2]][1],
               iz+vtx[vty[ic][2]][2]);
  }
  glEnd();
}
```

A degree of interactivity was provided by allowing keyboard presses to increase or decrease the mix percentage, in real time during the motion. This allows the operator to make the catch when the ball moves towards its envelope point. This gave a good feeling of catching (or missing) the ball.

The display included a representation of a football pitch and goalposts. However, this seemed inappropriate as the catch action was more suitable to a one-handed catch of a small ball rather than a football save.

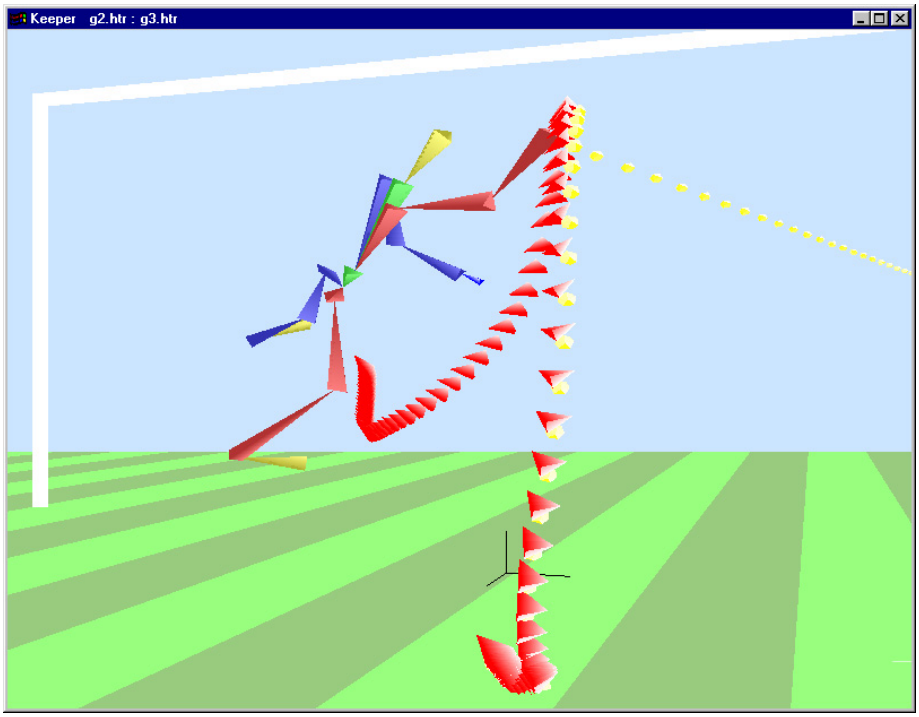


Figure 4.4: The Catch with Envelope Display

Chapter 5:Conclusions and Evaluation

Three main tasks have been undertaken during this project:

1. The producing of programs to display motion capture files.
2. The analysis of motion data types and mixing operations.
3. The producing of a simple interactive catch game.

5.1. Displaying Motion Capture Files

The requirements to visualise .bvh and .htr files were easily met using C++ and OpenGL. This gave good results in real time, including the ability to examine the motions from different angles. The OpenGL application is the industry standard as a cross platform graphics utility, and uses transformation functions which fit well with the motion capture data types.

Producing these programs gave the opportunity to analyse the way in which the hierarchical system works, particularly with respect to the use of base and frame transformations, which differed between the .bvh and .htr file formats with subtle implications for the resulting motion. The more complex .htr version gives more scope for rendering a realistic figure with lively movements.

5.2. Analysis of Data and Operations

Having built the display programs, attempts were made to produce interpolated results by mixing two different mocaps. Problems were encountered in attempting linear operations on the Euler angles. These can be summarised as:

1. Inconsistencies in the Euler representation of the original data.
2. Sudden `flips` as the angle passes through 180°.
3. The feet not making proper contact with the floor.
4. Attempts to mix asynchronous motions.

These problems mean that the motion capture sequences which are used must be very carefully chosen. When the original files are recorded in the motion capture studio the fact that an orientation is representable by different combinations of Euler angles is not relevant. Constraints to avoid these inconsistencies are often not included. When interpolations are performed these inconsistencies reappear, and are only solved through reinterpreting the data through trial and error.

Some motions, such as the example used in this project of animating a third hand equidistant between the minute and hour hand of a clock, are not solvable by

operations alone. They need the animator to make a conscious decision as to what happens when the element moves through 180°, for instance. Single axis data types such as quaternions and axis/angle pairs can actually harm this process. Because they arbitrarily limit the range of interpolations, they may miss the most appropriate result. However they do have advantages in speed and smoothness of interpolations.

There is considerable debate within the industry about the relative merits of these data types. In this project the most useful combination is to store the data as Euler angles, operate using quaternions and output the display as matrices.

The problem of the feet not touching the floor correctly in the mixed motion is not easily solved. Using inverse kinematics defeats the objectives of preserving the real-life motions, and is computationally costly. Trial and error selection of matching mocaps will minimise these errors.

This is also true with the use of asynchronous motions. The match needs to be close, especially in the bottom half of the body and when the figures jump in the air.

5.3. Production of the Catching Game

Two suitable mocaps were found which behaved well when mixed together, giving the motion of a diving catch to the top right and one to the bottom right. The interpolations are performed in real time, which means that it is possible to move from one motion to the other as the dive is being performed.

In order to produce the envelopes, a frame-by-frame set of 3D vertices is produced representing the position of the right hand, for each of the mixed sequences. The sets are stored as a 3D array of the triplet (vertex, frame, mix), at the start of the program.

As we now have a set of all possible positions of the right hand, an interactive game can be produced which compares the hand position to the ball position to see whether the catch has been made. In this project the possible ball positions were simplified so that the ball would always arrive at a position in the envelope. It is then a matter of the player`s skill to move the hand to the correct position to make the catch. Although this is a crude method it was felt that a more complicated algorithm was outside the scope of the project.

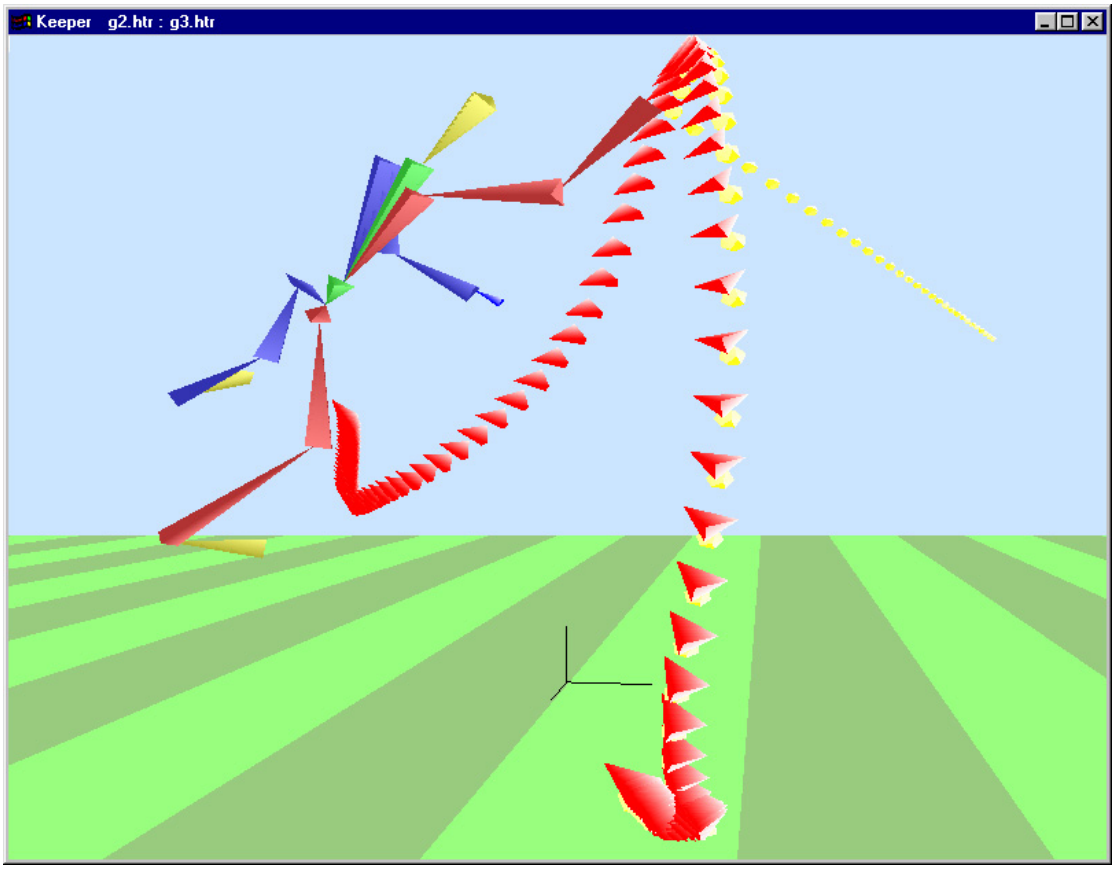
5.4. Evaluating the Results

At the start of the project it was not certain whether it would be possible to a) mix mocaps to give realistic motions and b) produce an envelope set which was usable in an interactive game. The final catch game gives encouraging results: the mixed

motions are subjectively appealing in the same way as the original `real` mocaps, and the interaction between player and ball via the envelope gives a fairly realistic and enjoyable sense of catching the ball.

The most difficult part of the process is finding suitable mocaps that will mix without the problems outlined in section 5.3. These problems would be alleviated if the original mocaps were recorded with interpolation in mind, by constraining the Euler angle data set so that the move into quaternion space can be negotiated safely. Even then, a number of sequences would be needed to produce, for instance, a complete set of goalkeeping moves. The jumping actions would need to be separate from the non-jumping ones (because of the need to conform to the laws of gravity). It should also be remembered that the hierarchical format is not symmetrical, so that right sided actions cannot be derived simply from left sided ones. Hence a full range of realistic moves would need at least eight original mocaps.

There is the need for much more research in all the project areas. As the leisure industry draws together the disparate areas of film, television, computer games and the web, one of the cementing forces will be the interactive use of real and manufactured action.



References

Books

James Arvo (1991) editor. Graphic Gems II. Academic Press Inc. 1991.

Alan Watt, Mark Watt (1992). Advanced Animation and Rendering Techniques. Addison-Wesley 1992.

Alan Watt (2000). 3D Computer Graphics (third edition). Addison-Wesley 2000.

Alberto Menache (1995). Understanding Motion Capture for Computer Animation and Video Games. Morgan Kaufmann 1995.

Mason Woo, Jackie Neider and Tom Davis (1997). The OpenGL Programming Guide. Addison-Wesley 1997.

Conference Papers

M. Gleicher (1999). Animation from Observation: *Motion Capture and Motion Editing.* Computer Graphics 33(4), p51-54. *Special Issue on Applications of Computer Vision to Computer Graphics.*

M. Gleicher (1997). Motion Editing with Spacetime Constraints. Proceedings of the 1997 Symposium on Interactive 3D Graphics.

M. Gleicher (1998). Retargetting Motion to New Characters. Proceedings of SIGGRAPH 98 Conference Proceedings, Annual Conference Series, p 33-42, 1998.

B. Bodenheimer, C. Rose, S. Rosenthal, and J. Pella (1997). The Process of Motion Capture: *Dealing with the Data*. Computer Animation and Simulation '97. Proceedings of the Eurographics Workshop.

Zoran Popovic, Andrew Witkin (1999). Physically Based Motion Transformation. SIGGRAPH 99, Los Angeles, August 8-13, 1999.

Jessica Hodgins and Nancy Pollard (1997). Adapting simulated behaviours for new characters. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings,* pages 153-162, August 1997.

Jonas Gomes, Luiz Velho, Fernando Wagner da Silva, Siome Klein Goldenstein (1999). Motion Processing Using Variable Harmonic Components. In Proceedings of SIBGRAPI'99, International Symposium of Computer Graphics and Image Processing, pages 49-58, Campinas, SP, Brazil, October 1999. IEEE Computer Society.

David J Surman (1997). A Brief History of Maotion Capture for Animation. Course Notes 9. *SIGGRAPH`97.*

Gordon Cameron (1997) (organizer). Motion Capture and CG Character Animation. SIGGRAPH '97 Panel.

Norman I. Badler et al (1999). Virtual human animation based on movement observation and cognitive behaviour models. Computer Animation Conf., Geneva, Switzerland, May 1999. (N. Badler, D. Chi, and S. Chopra)

Andrew Witkin and Michael Kass (1988). Spacetime constraints. In John Dill, editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 159–168, August 1988.

Peter C. Litwinowicz (1994). Inkwell: A 2½-D animation system. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH `91 Proceedings),* vol. 25, pps 35-42, July 1994.

Armin Bruderlin and Lance Williams (1995). Motion Signal Processing. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings),* Annual Conference Series pps 97-104, August 1995.

Charles F. Rose, III, Michael Cohen, Bobby Bodenheimer (1998). Verbs and Adverbs: Multidimensional Motion Interpolation IEEE Computer Graphics And Applications, Volume 18, Number 5.

Charles F. Rose III, Brian Guenter, Bobby Bodenheimer and Michael Cohen (1996). Efficient Generation of Motion Transitions using Spacetime Constraints *Proceedings of SIGGRAPH 96*.

Munetoshi Unuma, Ken Anjyo, and Ryozo Takaeuchi (1996). Fourier principles for emotion-based human figure animation. In *Proceedings of Computer Graphics (SIGGRAPH `96*), pages 75-84, 1996.

Theses/Dissertations

Michael Meredith (2001). Kinematics and Biomechanics in Adapting Motion Capture Data For Use in Computer Games. University of Sheffield Department of Computer Science 2001.

Michael Jones (2000). Mocap Merging: A Solution for Real-Time Modification of Motion Capture Data in a Games Environment. MSc Dissertation Project, University of Sheffield Department of Computer Science.

Stephen Dann (2001). Motion Capture Envelopes: *An Investigation into Real-time Modification of Motion Capture Data in a Game Environment*. University of Sheffield Department of Computer Science 2001.

Web References

Nick Bobick (2001). Rotating Objects using Quaternions. *www.gamasutra.com/features/19980703/quaternions_01.htm* CPM Media Inc. 2001.

Website of the Human Figure Animation Project – *Microsoft Research Graphics Group*. www.research.microsoft.com/Graphics/HFAP/

Appendix 1 Project Diary

18th June	Installation of Java software and collection of mocaps
25th June	Java-based .htr and .bvh programs completed
2nd July	Installation of C++ and OpenGL
23rd July	Final .htr and .bvh programs produced in C++
10th August	Completion of analysis of data types and operations
24th August	Completion of interactive catching game
30th August	Completion of written dissertation

Appendix 2 Programs Produced for Project

BVHPlayer.java
Inputs a .bvh file and displays the motion in the three planes X-Y, Y-Z, and X-Z.

BVHPrint.java
Inputs a .bvh file and prints out the vertices of the left foot for each frame.

BVHQ.cpp
Inputs a .bvh file and three angle parameters and displays the motion. Interpolation is undertaken using quaternions.

BVHTrace.java
Inputs a .bvh file and displays the motion of each frame superimposed on the screen, giving a trace of the whole motion.

D3.cpp
Inputs a .htr file and three angle parameters and displays the motion. The action can be viewed from different angles, and the frame rate changed.

Envelope.cpp
Inputs two .htr files, a mix percentage and a frame offset between the files. An envelope of the right and left hand are displayed. The viewing angle can be changed.

EulerTest.java
Inputs three Euler angles and prints out the resulting coordinate positions, given a Y translation of 10.

EulerTest180.java
Inputs three Euler angles and prints out the resulting coordinate positions. Given a Y translation of 10. Repeats the process to verify that ry+180, 180-rx, rz-180 gives the same result.

EulerToPoints.java
Inputs three Euler angles and three translation values prints out the resulting coordinate positions.

Half.java
Inputs a .bvh file and a new filename. Produces a new .bvh file where all angles are divided by two.

HTR2.java
Inputs a .htr file and displays the motion in the three planes X-Y, Y-Z, and X-Z.

HTRMixer.cpp
Inputs two .htr files and displays the resulting mixed motion. In real time, the viewing angle, the mix percentage, the speed and a trace of the envelope can all be varied.

HTRQ.cpp
Repeats the functions of **HTRMixer.cpp.** The data is converted into quaternions for the interpolation operations and then converted to matrices.

Keeper.cpp
The final interactive game. This plays a mixed combination of mocaps **g2.htr** and **g3.htr**, which must be included in the directory. At the key press a ball moves towards the goal and the operator presses to keys to move between the two mocaps to try to catch the ball. An automatic save is also provided.