Interactive Synthesis of Avatar Motion from Preprocessed Motion Data

Submitted in Partial Fulfilment for a Bachelor of Science (Honours) Degree at

RHODES UNIVERSITY Where leaders learn

by

Andrew Simon Peirson

Supervisors: Professor Shaun Bangay, Professor George Wells and Adele Lobb

10 November 2003
Abstract

A recent dilemma being faced by animators, especially within the context of three-dimensional games and virtual environments, is the ability to produce natural avatar motion in real-time. Motion capture is one of the most effective methods of animating realistic human motion. However, one drawback of the motion capture process is the fact that the range of avatar movement is limited to the set of avatar behaviours that have already been produced. Thus if an existing motion is not appropriate for a particular context, the motion capture process would have to be repeated. Since motion capture is a fairly time consuming process, it is not surprising that a number of research efforts have focused on editing already captured motion sequences.

The aim of this project is to research and implement an approach, based on a paper by Lee et al (2002), to synthesizing new motion for a controllable avatar from a set of preprocessed motion data. This paper discusses the issues surrounding the implementation of the chosen approach and how it differs from the approach used by Lee et al (2002).

During implementation of the above approach, a motion archive was compiled from a number of sources on the Internet and the stored raw data was used to test the effectiveness of the motion synthesis process. It was concluded that the developed motion synthesis and display tool is relatively successful in blending different motion files to produce new motion sequences, but the results could be improved by assembling a larger, less diverse motion archive.
There are a number of people who have helped me achieve my goals and ambitions during my time at Rhodes University and I would like to thank them, especially for their support and guidance towards the completion of this thesis and related project.

Firstly, I am extremely grateful to my supervisors, Professor Shaun Bangay, Adele Lobb and Professor George Wells, for the assistance and encouragement they have given me throughout this year. This research effort would not have been possible without their constant support which came in the form of numerous ideas, advice and constructive feedback. Special thanks also go to my supervisors who acted as my group of proofreaders.

Also, my research efforts would have been pointless without raw data to test the tools developed. To this end, I am extremely grateful to those people who made their collected motion files freely available for anyone to download off the Internet. A full this of the sources used to compile the motion archive is given in Appendix C towards the end of this thesis.

During my last few years at Rhodes University, I have met an awesome bunch of people and have made many good friends along the way. I would like to take this opportunity to extend my appreciation to them all. Firstly to the Goldfield’s guys – shot for always keeping me smiling. To my friends and colleagues in the Honours Labs I am extremely grateful for all the laughs, support and companionship – you guys rock!

Last, but not least, I would like to thank my family for the support and encouragement they have given me during my time at Rhodes and before. Special thanks go to my parents for giving me the opportunity to study at an awesome university … I am truly blessed.
# Table of Contents

Abstract ........................................................................................................... i

Acknowledgements ........................................................................................ ii

1 Introduction ...................................................................................................... 1
  1.1 Context ........................................................................................................ 1
  1.2 Avatar Animation and Control ..................................................................... 1
  1.3 Techniques for Obtaining Realistic Motion ................................................ 3
    1.3.1 Keyframe Animation ............................................................................. 3
    1.3.2 Physically Based Modelling ................................................................. 4
    1.3.3 Motion Capture .................................................................................... 4
  1.4 Motivation ..................................................................................................... 6
  1.5 Project Aim and Objectives ......................................................................... 7
  1.6 Thesis Overview and Layout ..................................................................... 8

2 Background and Related Work ....................................................................... 10
  2.1 Discussion of Foundation Papers ................................................................ 10
    2.1.1 Interactive Motion Generation from Examples ..................................... 10
    2.1.2 Interactive Control of Avatars Animated with Motion Data .................. 13
  2.2 Similar Research Efforts ............................................................................. 16
    2.2.1 Blending Existing Motion Data .............................................................. 16
    2.2.2 Constraint-Based Motion Editing ......................................................... 17
    2.2.3 Retargeting Motion to New Characters ............................................... 18
  2.3 Motion Capture File Formats .................................................................... 18
    2.3.1 Acclaim Data (ASF/AMC) ................................................................... 20
    2.3.2 BioVision Data (BVH) ......................................................................... 20
    2.3.3 Motion Analysis Data (HTR) ................................................................. 21
  2.4 Summary ..................................................................................................... 21

3 Design ............................................................................................................. 23
  3.1 System Requirements ................................................................................. 23
  3.2 Design Issues ............................................................................................. 23
    3.2.1 Choice of Motion File Format ............................................................... 24
    3.2.2 The Need for a Motion File Conversion Tool ....................................... 25
    3.2.3 Design of the Maximal Skeleton Hierarchy ......................................... 26
    3.2.4 Choice of the User Interaction Technique .......................................... 28
  3.3 System Architecture .................................................................................. 29
  3.4 Motion File Converter Design ................................................................... 30
    3.4.1 Module Requirements .......................................................................... 30
    3.4.2 Interface Design .................................................................................. 31
  3.5 Motion Display and Synthesis Tool Design ................................................. 31
    3.5.1 Module Requirements .......................................................................... 32
    3.5.2 Interface Design .................................................................................. 32
  3.6 Design of the Data Structures Used ........................................................... 33
    3.6.1 Terminology ....................................................................................... 34
    3.6.2 A Sample BVH File ............................................................................. 34
# Table of Figures

Figure 1-1: An example of an avatar illustrating a typical multi-joint character. ........ 2
Figure 1-2: Keyframe animation of a computer model........................................... 3
Figure 2-1: Motion Sequences Represented by a Directed Graph ...................... 11
Figure 2-2: Joint Constraints Leading to a Jumping Motion............................... 12
Figure 2-3: Motion Data Preprocessing............................................................... 14
Figure 2-4: Interfacing and Avatar Control Techniques .................................... 15
Figure 2-5: Example of Retargeting Motion to New Characters......................... 18
Figure 3-1: BVH File Comparison Based the Skeleton Hierarchy ....................... 25
Figure 3-2: Maximal Skeleton Hierarchy............................................................ 27
Figure 3-3: Graphical representation of the Maximal Skeleton ......................... 28
Figure 3-4: System Architecture illustrating individual System Components ....... 29
Figure 3-5: Proposed User Interface for Motion File Converter ......................... 31
Figure 3-6: Proposed User Interface for Motion Display and Synthesis Tool ....... 33
Figure 3-7: Proposed Menu Options ................................................................. 33
Figure 3-8: UML Description of the Problem Domain Classes ......................... 39
Figure 3-9: UML Description of the User Interface Classes.............................. 40
Figure 4-1: Skeleton Object Used For Rendering ............................................. 53
Figure 4-2: Graphs plotting Equation 1 based on different values for \( \sigma \) ......... 55
Figure 4-3: Motion Synthesis Process................................................................. 59
Figure 5-1: Conversion Technique Dialog.......................................................... 66
Figure 5-2: Screen Shot of the BVH Converter Tool ......................................... 67
Figure 5-3: Error in Solid Rendering .................................................................. 68
Figure 5-4: User Interface of the Motion Display and Synthesis Tool ............... 70
Figure 5-5: Dialog Used to Request Markov Process Settings ......................... 71
Figure 5-6: Incompatible Skeleton Hierarchy Error Message ......................... 71
Figure 5-7: Two Consecutive Frames Illustrating a Transition Point ................. 72
Figure 5-8: Using the Choose Transitions Option ............................................. 73
Figure 5-9: Two Consecutive Frames without Interpolation ................................. 74
Figure 5-10: Using Interpolation to Blend Transitions ....................................... 75
Figure 5-11: Exported Motion in Blender ............................................................ 75
Figure 5-12: Screen shot illustrating possible Graph Types............................... 76
Figure 5-13: Graph Options.................................................................................. 76
Figure 5-14: Graph comparing motion files based on their Possible Transitions .... 77
Figure 5-15: Graph using Bezier Curve Style .................................................... 78
Figure 5-16: Main Transition Jumps between dribble and dunk Motions ........... 79
Figure 5-17: Sample Orientation values for the Root Joint ............................... 80

Index ....................................................................................................................... 85
References.............................................................................................................. 86
Appendix A – Shell Script to Generate HTML .................................................. 90
Appendix B – Project Source Code .................................................................. 92
Appendix C – Source List .................................................................................... 92
List of Tables

Table 2-1: Common Motion Capture File Formats ................................................................. 19
Table 3-1: Source References and File Names of the BVH Files Compared ....................... 25
Table 3-2: Synonyms of Maximal Skeleton Joint Names ...................................................... 26
Table 3-3: A 2D array representing a Motion segment of a BVH file ............................... 39
Table 4-1: Basic Operations Using Quaternions ................................................................. 47

Table of Listings

Listing 3-1: Sample BVH File (from Palmer, 2001) ............................................................. 35
Listing 3-2: Main Attributes and Operations of the SkeletonNode Class ....................... 37
Listing 3-3: Main Attributes and Operations of the Skeleton Class ................................. 37
Listing 3-4: Main Attributes and Operations of the Channel Class ................................. 38
Listing 3-5: Main Attributes and Operations of the Movements Class ............................. 38
Listing 4-1: Shell Script Snippet showing the creation of the Category Links ................. 45
Listing 4-2: Shell Script Snippet showing BVH Table Generation ................................. 45
Listing 4-3: Setting up a map to store the Joint Synonyms ............................................. 48
Listing 4-4: Code Snippet showing part of the map Initialisation .................................. 48
Listing 4-5: Code Snippet showing how the Skeleton Joints are renamed ....................... 48
Listing 4-6: Calculating the change needed in Orientation Values for each joint .......... 50
Listing 4-7: Updating Bone Lengths .................................................................................. 51
Listing 4-8: Method to Calculate Velocity Differences .................................................... 57
Listing 4-9: Calculating the First term of the Position Differences Function ................... 57
Listing 4-10: Calculating the Second term of the Position Differences Function ............ 58
Listing 4-11: Vector List representing Results of the Markov Process ........................... 60
Listing 4-12: Vector List representing the current Motion Database .............................. 61
Listing 4-13: System call to Open Blender ........................................................................ 64
1 Introduction

1.1 Context

According to Arikan and Forsyth (2002), motion is one of the key elements of all computer graphics endeavours. Thus, there are a multitude of applications that require large quantities of realistic looking motion. One of the most important uses of natural looking, computer generated motion is within the three-dimensional computer gaming and movie industry, where large repertoires of actions are required to produce realistic motion for human characters in particular. Due to the popularity of such applications, it has become apparent that the synthesis of novel, natural human motion is an important problem being faced by animators worldwide. However, as explained in the next section, there are a number of difficulties involved with the production of natural looking motion, especially when it is people who must move.

1.2 Avatar Animation and Control

As mentioned above, a new class of interactive applications involving 3D animated characters has become popular, mainly due to major advances in computer hardware, software and networking technologies. Animated characters can play several roles within a virtual environment, but Brogan, Metoyer and Hodgins (1998) state that all characters must at least have a variety of complex and interesting behaviours and must be responsive to the user’s actions.

Kuffner (1999) identifies two additional characteristics that all animated characters possess: the degree of autonomy and the degree of interactivity. These characteristics are directly related to the field in which the animated character is being used. The degree of autonomy refers to the extent of user intervention required by the character in order to perform a certain task. At one end of the scale are user-controlled characters that require individual joint motions to be defined by hand. At the other end of the scale are fully autonomous characters that require no user intervention as their motions are defined by algorithms. The degree of interactivity refers to response time constraints based on the
application domain. Non-interactive applications do not require real-time response rates to user interaction whereas highly-interactive applications need to maintain display update rates of about 30 frames per second (Kuffner, 1999). Variations in the degree of these two characteristics bring about several potential uses for character motion generation software that covers a wide range of applications.

A principal subset of these 3D animated characters is that of **avatars**. According to Vilhjálmsson (1996), the term *avatar* is used to refer to “any graphical representation of a *human* in a multi-user *computer* generated environment”. Due to the complex, multi-jointed structure of human character models, avatar animation poses particular challenges to motion generation. According to Pullen and Bregler (2002), a typical articulated figure model such as an avatar usually has at least 19 separate joints and 63 degrees of freedom. Each joint has 3 degrees of freedom (rotation about the x, y and z axis) and the root joint has an additional 3 degrees of freedom relating to translational data. An example of such a model can be seen in Figure 1-1.

Therefore, because of the complex structure of inter-related joints that describe avatars, the process of designing a rich set of natural looking behaviours for them is laborious, time consuming and requires extensive expertise. The next section identifies the main methods used by animators to produce realistic avatar motion.

![Figure 1-1: An example of an avatar illustrating a typical multi-joint character.](image-url)
1.3 Techniques for Obtaining Realistic Motion

Arikan and Forsyth (2002) describe three techniques that are used to obtain natural looking motion, namely \textit{keyframe animation}, \textit{physically based modelling} and \textit{motion capture}. The choice of which method is appropriate to use depends on the specific situation in which the animation is being produced. Although this thesis mainly refers to the use of motion capture data, it is still important to examine each animation technique in terms of their relative advantages and disadvantages.

1.3.1 Keyframe Animation

With key framing, a 3D model of a character is created in the computer, and the animator specifies \textit{keyframes} (critical or key positions) for the animated object by posing the model in the computer. The computer then fills in the missing frames by interpolating between these positions. This process is illustrated in Figure 1-2.

![Keyframe animation of a computer model](from Pullen and Bregler, 2002)

In the above example, the animator specifies the start pose of the avatar (the top, left frame) and the final position of the avatar (the top, right frame) and the computer generates the in-between frames. The graph represents the same process by plotting the avatar’s spine angle as a function of time.
Using traditional key framing, it is relatively straightforward to construct motion for rigid objects through translational and rotational trajectory curves. However, manipulating and coordinating the limbs of a human figure via key frames is a complex task that requires lots of work and highly developed skills. The animator must possess a thorough understanding of how moving objects should behave over time as well as the skill to represent those movements. Another problem with computer based keyframe animation is that the interpolation process only generates smooth spline curves or lines, similar to the one shown in Figure 1-2, for the in-between frames. This lack of variation may result in motion that does not represent the way a live human moves (Pullen and Bregler, 2002).

1.3.2 Physically Based Modelling

Otte (2000) describes physically based modelling as the process of modelling the laws of physics and how they affect the motions of bodies. Thus, given initial positions, velocities, forces, and dynamic properties, an object's motion is simulated according to natural physical laws. One advantage of this method is that it frees the animator from having to worry about low level details of the motion while ensuring that the motion is physically realistic. However, this is done at the cost of having less control over the movements (Yu and Terzopoulos, 1998). Also, since motion is produced on the fly using control algorithms, this technique of animation is better suited to highly interactive applications such as video games and virtual environments. Once again, this method works well when applied to simple systems (such as animating falling objects, smoke, wind, water, and other natural effects), but is difficult to use with more complex arrangements such as the human body. Due to the complex dynamics of a multi-joined system, designing an accurate control algorithm to model the forces and torques involved in producing a particular motion is an extremely difficult task (Kuffner, 1999). Brogan et al (1998) also state that computational cost is a major disadvantage of physically based modelling.

1.3.3 Motion Capture

An alternative method to obtain realistic avatar motion is motion capture. According to Arikan and Forsyth (2002), motion capture is a “standard solution” to some of the
drawbacks of the other two animation methods. Real-time 3D motion capture equipment comprising of magnetic or vision based sensors is used to record motion data for an approximate skeletal hierarchy of a live subject. This motion capture data can be used to facilitate conventional animation, or to create libraries of reusable clip-motion which serve as databases for on-the-fly assembly of animation in interactive systems. (Kim, 2000) Due to the relative ease of recording many human actions, motion capture is becoming extremely popular. This is especially true within the domain of sports video games, where motion capture is used to create natural looking movements of athletes within virtual environments (Brogan et al, 1998). Another advantage of the motion capture technique, identified by Pullen and Bregler (2002), is that it immediately provides motion data for all degrees of freedom at a very high level of detail.

However, a number of drawbacks preclude motion capture from being an ideal solution for all applications. The major problems associated with motion capture are discussed next:

- Arikan and Forsyth (2002) make the point that most motion capture systems are very expensive to use since the process is time consuming and in the end motion capture data tends not to be reused.
- Also, some types of motion capture equipment (such as magnetic systems) require cables to be connected to the sensors on the subject. This restricts the actor’s range of motion, thereby making it difficult to record certain movements.
- Lastly, the captured data may not be exactly what the animator wants. (Pullen and Bregler, 2002) If this is the case, the motion capture session would have to be repeated until the desired motion is achieved. Such repetition can be costly and labour intensive.

The last drawback mentioned above is a major problem to animators worldwide, mainly because it is often difficult to know exactly what motions are desired before the motion capture session begins. Therefore, as a result, recent research efforts in this field have been aimed at developing methods for editing and processing the motion capture data after it has been collected in order to make it re-usable. This argument forms the reasoning behind the motivation for this project, as explained next.
1.4 Motivation

If motion capture data represents the movements of real human actors, one may ask why it is necessary to edit that data once it has been captured. Gleicher (1999) states that a common misconception is that the importance of motion editing for motion capture stems from the fact that motion capture data is imperfect. However, even when motion capture data represents a desired motion, there are still reasons why it’s often necessary to make modifications to the motion. Gleicher (1999) provides the following reasons:

- **Re-use**: captured motion data represents a unique character structure performing a particular action. If it is necessary to apply this motion to several different characters, or adjust their behaviours slightly, the motion data must be edited.

- **Creating Infeasible Motions**: it would be impractical to film a motion capture scene incorporating injury or destruction of the character, as this would entail the same effects occurring to the actor. Thus there is a need to be able to edit the motion.

- **Imperfections of reality**: not all actions are perfect every time. This is especially true with repetitive behaviours where each repeated cycle will differ slightly from the last. Editing tools could then be used to introduce a bit of variation into each consecutive cycle to get a more realistic looking motion.

Kim (2000) gives a good reason motivating the need for motion editing tools and related research by stating “In the absence of effective editing tools, a recorded movement that is not quite “right” requires the whole data capture process to be repeated. So, it is desirable to develop tools that make it easy to reuse and adapt existing motion data.”

Therefore, as the above authors have pointed out, there is a great need to research techniques that could be used to get extra use from already collected motion capture data. Chapman (2000) mentions that most research concentrates on combining motion captures, constraining movement in some way, the use of space-time constraints or some other modification technique. Research has also focussed on developing tools that are capable of retargeting a motion from one character to another or from one environment to another (Lee and Shin, 1999). These various motion editing techniques will be briefly discussed in Chapter 2 “Background and Related Work” under Section 2.2, “Similar Research Efforts”.
Thus, the goal of this project is to investigate the implementation of a simple tool that could be used to analyse and adapt existing motion capture data so that new realistic motions can be synthesized without the need for an additional motion capture session.

1.5 Project Aim and Objectives

It is the aim of this project to research and implement a solution to the previously mentioned drawbacks of using motion capture data to produce avatar animation. The solution adopted is largely based on the technique described by Lee et al (2002) in their paper entitled “Interactive control of avatars animated with human motion data”. While many video games today make use of a large motion database that consists of many short, carefully planned, labelled motion clips to provide natural avatar motion, Lee et al (2002) describe an alternative method which makes use of a pre-processed motion database of longer, unstructured sequences of motion capture data that allows the animator to synthesize new avatar motion based on simple input. In broad terms, their approach describes an algorithm that compares different motion capture files and identifies possible ways of generating new human motion by cutting and pasting existing motion capture data. An in-depth discussion of how the motion files are compared is given in Chapter 2 “Background and Related Work”, under Section 2.1 “Discussion of Foundation Papers” and in Chapter 4 “Implementation”. I have identified four main project objectives or goals which are now discussed in greater detail below:

**Objective 1:** Create an archive of motion capture data in a common format. At present, there are many different types of motion capture files available, each slightly different in terms of the way the skeleton hierarchy and the actual motion data relating to the various joints of the skeleton is stored. To compare different motion capture files, it is necessary to build up a large database of motion capture data of the same format. This implies that a single file format must be chosen and used throughout the project. The various motion capture file formats available are evaluated and discussed further in Chapter 2 under Section 2.3 entitled “Motion Capture File Formats”.
**Objective 2:** *Develop a tool to animate the motion data using OpenGL.* The tool must be able to read motion capture files of the chosen format and then render an avatar based on the motion data stored in the file.

**Objective 3:** *Investigate and implement an approach to synthesize new motion sequences from existing motion data given some simple user input.* In addition to displaying the avatar motion in a 3D environment, the tool should allow the animator to generate and store new motion sequences based on some simple input.

**Objective 4:** *Integrate this technique for use with the open source, 3D modelling and animation environment, Blender.* In order to make the tool more useful, it should allow for easy integration with an external 3D environment such as Blender. If the results of using the motion editing tool could be exported to Blender, additional modelling and skinning could be performed more easily which would improve the end product.

### 1.6 Thesis Overview and Layout

The sections detailed above have established that there is extensive research being done that is focussed on developing techniques to process the captured motion data. The remainder of this thesis is divided into five chapters, structured as follows:

**Chapter 2:** reviews background information and related work. The main section in this chapter discusses the foundation papers of Lee et al (2002) and Arik and Forsyth (2002). Similar research efforts are also mentioned and their relevance to this thesis identified. Lastly, the various motion capture file formats in use today are discussed based on their relative merits and weaknesses.

**Chapter 3:** relates to the design of the various tools that are developed during implementation. Firstly, an overview of the entire system architecture and requirements is given. Next, the design of the user interface used for the tools is reviewed. Finally, a description of the data structures that are used by the tools is given.
Chapter 4: discusses the implementation of the designs from Chapter 3. The chosen implementation languages and platforms are identified and the choices made are motivated and explained. The rest of the chapter describes the implementation issues that relate to each of the main project objectives already identified in Section 1.5.

Chapter 5: summarizes the results that have been achieved relative to the project objectives as well as results achieved from additional experiments. It discusses the functionality of each tool developed and provides a number of screen shots of the working application to illustrate the effectiveness of the system.

Chapter 6: summarizes and discusses the entire research effort of the project. The contributions and conclusions of the work are identified. Some of the difficulties uncovered during this project and ways to address them are also discussed. Lastly, approaches for future improvements and future work are suggested.
2 Background and Related Work

According to Lee et al (2002), “motion capture data is the most common technique in commercial systems because many of the subtle details of human motion are naturally present in the data rather than having to be introduced via domain knowledge.” The sudden increase in popularity of using motion capture data for animating 3D characters and articulated figures has led to a tremendous amount of research on developing methods for editing already captured motion data. The bulk of this chapter focuses on the theory behind the motion synthesis and editing techniques discussed in the papers1 by Lee et al (2002) and Arikan and Forsyth (2002). Alternative motion generation and editing methods are then briefly discussed so as to put the project in context and to establish the origins of any related work. Finally, a concise summary and evaluation of the different motion capture file formats in use today is given, so that design choices and implementation issues can be motivated and explained at a later stage.

2.1 Discussion of Foundation Papers

To begin with, it is important to note that the implementation methodology that I chose to follow relates more closely to the techniques described by Lee et al (2002), rather than those outlined by Arikan and Forsyth (2002). For this reason, I will only briefly discuss the paper by Arikan and Forsyth (2002) and then explore in greater detail the procedures adopted by Lee et al (2002).

2.1.1 Interactive Motion Generation from Examples

In their paper, Arikan and Forsyth (2002) present a “framework that generates human motions by cutting and pasting motion capture data”. The underpinning of their method is based on the construction and employment of a “motion graph” that they use to represent the collection of possible motion sequences obtained by comparing the motion files within their motion dataset. As shown in Figure 2-1, each possible motion sequence is

1 “Interactive Control of Avatars Animated with Human Motion Data” (Lee et al, 2002) and “Interactive Motion Generation from Examples” (Arikan and Forsyth)
represented in the graph as a node. Also, there is an edge between nodes for every frame in one sequence that can be spliced to a frame in another sequence. A valid path in this graph therefore represents a collection of splices between frames that could be applied in order to generate a new motion. Selecting a combination of clips that yields a realistic motion is therefore done by “searching appropriate paths in the motion graph using a randomized search method.” (Arikan and Forsyth, 2002)

![Figure 2-1: Motion Sequences Represented by a Directed Graph (from Arikan and Forsyth, 2002)](image)

Attached to each edge in the graph is an associated cost value that represents the degree of fitness of transitioning between the two involved frames. For example, if splicing from one sequence to another along an edge introduces a discontinuous motion, then the cost attached to the edge is high. The cost function used by Arikan and Forsyth (2002) takes into account the difference between joint positions and velocities for the two frames as well as the difference between torso velocities and accelerations. This cost weighting system is used to discover the most appropriate path to follow in the motion graph, in order to produce realistic motion that satisfies user defined constraints.
It becomes obvious at this stage that as more and more motion files are added to the motion graph, the number of nodes and edges increases rapidly. Also, due to the fact that for every valid splice between, say frame \( i \) in Motion 1 and \( j \) in Motion 2, there will more than likely be a similarly valid splice between their preceding and succeeding frames. These two facts imply that the graph could contain a number of equally valid paths that would satisfy the user defined constraints and therefore searching the graph becomes a complex problem. To get around this issue, Arikan and Forsyth (2002) develop a method of summarizing the motion graph which they call \textit{clustering}. Clustering involves compressing blocks of similar node-edge relationships in order to reduce the number of paths that could be chosen from the motion graph. Lee et al (2002) also make use of a similar form of cluster analysis which will be described later.

A form of genetic algorithm is used to search the summarized motion graph to generate different alternative motions through the use of mutations. The search algorithm is repeated until no better path can be generated. In other words, the first mutation pass replaces a motion sequence with a better one and the second mutation pass adjusts the finer details of the motion.

Figure 2-2 shows a screen shot from the tool developed by Arikan and Forsyth (2002). It illustrates how constraints are considered when searching the motion graph to generate new motions. In this case, the head of the avatar is constrained to be high in the position indicated by the blue line. This constraint leads to a jumping motion being chosen when the motion graph is searched for paths to possible motion sequences.

\textbf{Figure 2-2:} Joint Constraints Leading to a Jumping Motion (from Arikan and Forsyth, 2002)
According to Arikan and Forsyth (2002), their framework generates motion sequences that satisfy a variety of constraints, are smooth and natural-looking and are generated in real time using interactive authoring, thereby allowing widespread re-use of motion capture data for new purposes.

### 2.1.2 Interactive Control of Avatars Animated with Human Motion Data

Lee et al (2002) discuss an alternative method of motion synthesis to that of Arikan and Forsyth (2002). Instead of a motion graph, Lee et al (2002) make use of a motion database that stores extended, unlabeled sequences of motion capture data pertinent to the application domain. They discuss how raw motion data from the underlying motion dataset can be “preprocessed for flexibility and efficient search and exploited for real-time avatar control using various interface techniques”.

Firstly, the underlying motion data is preprocessed to add variety by identifying likely transitions between motion segments based on good matches in poses, velocities and contact states of the character. Searching the resulting unstructured motion data structure in real time is a major challenge, so they then summarize the motion data using a form of cluster analysis.

Due to the two step process described above, the data representation of the human motion suggested by Lee et al (2002) consists of a two-layer structure. “The higher layer is a statistical model that provides support for the user interfaces by clustering the data to capture similarities among character states. The lower layer is a Markov process that creates new motion sequences by selecting transitions between motion frames based on the high-level directions of the user.” (Lee et al, 2002)

The result of the Markov process is represented as a matrix of probabilities \( P_{ij} \) describing the likelihood of transitioning from frame \( i \) to frame \( j \). The function used to estimate \( P_{ij} \) will be discussed in more detail in Chapter 4 “Implementation” and the issues involved with its implementation in the motion synthesis tool will be identified. What is important to realize at this stage is the fact that the matrix of probabilities is dense and requires large
amounts of storage space. Lee et al (2002) therefore outline four rules that they use to prune the database. These are:

- Transitions between motion segments with dissimilar contact states are removed.
- All probabilities below a user-specified threshold are set to zero.
- The best transition among many similar transitions is chosen by selecting the local maxima in the transition matrix and then setting the probabilities of the other transitions to zero.
- Probabilities of transitions that result in dead ends (that lead to a portion of motion that has no exits) after a few hops, are set to zero.

As already mentioned, after applying the Markov model, the resulting data structure is too complex for efficient search or clear presentation in a user interface, so Lee et al (2002) generalize the data using a form of cluster analysis which captures the distribution of frames and transitions in a summarized manner. Similar motion frames are grouped together in “clusters”. The collection of possible transitions at any given motion frame is stored in a data structure known as a “cluster tree”. Lee et al (2002) give the name “cluster forest” to the collection of cluster trees that forms the higher layer. These concepts are illustrated in Figure 2-3.

Figure 2-3: Motion Data Preprocessing (from Lee et al, 2002)
Part (A) of the diagram represents the initial state of the motion database before any processing occurs. Each coloured arrow corresponds to a different motion clip made up of a number of frames. In part (B), the Markov model is applied and the resulting data structure consists of all the possible frame-to-frame transitions. Part (C) shows the results of grouping similar frames into clusters. Part (D) illustrates the creation process of cluster trees. The motion database is traversed and a new node is added to the cluster tree each time a cluster boundary is crossed.

Finally, Lee et al (2002) discuss how their preprocessed motion database can be used to generate new motions and control an avatar. They demonstrate three interface techniques:

- The user selects from a set of available choices that are appropriate to the avatar’s environment. An example of this technique in use is shown by the bottom left screen shot in Figure 2-4 below.
- The user sketches a path through an environment using the mouse. The top left and top right screen shots in Figure 2-4 illustrate the use of sketching as an avatar control method.
- The user acts out a desired motion in front of a video camera. A silhouette of the human body is constructed from the video and used to determine avatar motion. This technique is shown in the bottom right screen shot of Figure 2-4.

Figure 2-4: Interfacing and Avatar Control Techniques (from Lee et al, 2002)
In conclusion, Lee et al (2002) present a framework for preprocessing raw motion capture data to construct a motion database that can be used to generate new motion. Statistical techniques such as the Markov model and cluster analysis are used to build up a two layer structure that forms the motion database. Three different interface techniques are also presented that allow the user to control an avatar in real-time.

2.2 Similar Research Efforts

According to Arikан and Forsyth (2002), motion synthesis follows two main schools of thought: motion generated by modifying existing motion data and motion generated through the use of controllers\(^2\). However, since the focus of this thesis is on motion synthesis from examples, the research related to controller based approaches is not reviewed. Instead, research related to motion editing and adaptation techniques will be discussed. Most of these research efforts concentrate on methods to re-use motion capture data by blending existing motion capture data, by constraining movement in some way or through the use of space-time constraints.

2.2.1 Blending Existing Motion Data

Lee et al (2002) and Arikан and Forsyth (2002) are not the only researchers who have developed systems that generate novel motion from combining existing motion data.

Lamouret and van de Panne (1996) have developed a system that dynamically generates a hopping motion for a simple “Luxo” lamp depending on the pose of the character, the local terrain and any user preferences that have been specified. Their method differs from the other techniques discussed so far in that the underlying motion database is made up of motion samples generated using physics-based animation, not from motion capture data. Also, the character they have used to implement their techniques is very simple compared to that of a human character.

---

\(^2\) Controller based approaches use physical models of systems and algorithms to produce output in the form of forces and torques that can be used to estimate avatar movements (Arikан and Forsyth, 2002).
Wiley and Hahn (1997) describe a process of creating new motions based on user specifications by mixing motions selected from a motion database. In this case, the raw example motions can be generated by key framing, physical simulations or motion capture. To blend simple motion data segments, Wiley and Hahn (1997) use a form of linear interpolation.

Kovar, Gleicher and Pighin (2002) discuss how a motion graph, similar to the one described by Arikan and Forsyth (2002), can be used to generate new motion by “building walks³ on the graph”. Their framework discusses how particular walks that satisfy user requirements can be extracted from the motion graph. The distance measure that Kovar et al (2002) use to identify matching transitions in the motion graph is similar to that used by Lee et al (2002), first introduced by Schodl et al (2000).

2.2.2 Constraint-Based Motion Editing

According to Gleicher (2001), “constraint-based motion editing makes some of the features that are to be preserved or changed in a motion explicit as constraints”. A number of recent motion synthesis research endeavours have focused mainly on constraint based motion adaptation, which can be used to compute changes to motion data so that specific user needs are met. Litwinowicz and Gleicher (1998) combine this type of motion synthesis with a technique known as motion-signal processing which ensures that the newly generated motions “meet user goals while retaining much of their original quality”.

Another form of constraint-based motion editing makes use of spacetime constraints. According to Witkin and Kass (1988), spacetime constraints allow an animator to specify what the character must do, how the motion should be performed, the character’s physical structure and the physical resources available for the character to use to accomplish the motion. Gleicher (1997) specifies how spacetime constraints differ from other constraint methods by stating that spacetime constraints consider the entire motion simultaneously, rather than motion at an individual frame. This allows the user to interactively position characters using direct manipulation. Other researchers that make use of spacetime constraints for motion editing and synthesis include Rose et al (1996) and Liu et al (1994).

³ A graph walk is a particular sequence of nodes of the motion graph.
2.2.3 Retargeting Motion to New Characters

Another area of research that centres on the modification of existing motion data is that of adapting an animated motion from one character to another. Gleicher (1998) describes a method that focuses on adapting the motion of one articulated figure to another figure with identical structure but different segment lengths. His approach also makes use of a spacetime solver that preserves specific features of the original motion. For example, the fact that a character’s feet must touch the floor when walking is identified as a constraint. If the newly retargeted motion violates this constraint then the motion is modified slightly until the constraint is satisfied.

Figure 2-5 above shows how Gleicher (1998) was able to adapt a walking motion to a number of differently proportioned figures. The left hand figure represents the original actor. The smaller actor (60% of the size of the original figure) is forced to use the original foot plant positions.

2.3 Motion Capture File Formats

Due to the success of motion capture, a number of companies have been established that can record and provide motion data. Unfortunately, most of these production houses\(^4\) have developed their own file format which has led to a lack of standardization. Table 2-1

below shows the major motion capture file formats in use today. The table also identifies the companies that are associated with each file format and where additional information can be obtained about the specifications of each of the formats.

<table>
<thead>
<tr>
<th>File Extension</th>
<th>Associated Company / Description</th>
<th>File Format Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASF &amp; AMC</td>
<td>Acclaim</td>
<td><a href="http://www.darwin3d.com/gamedev/acclaim.zip">www.darwin3d.com/gamedev/acclaim.zip</a></td>
</tr>
<tr>
<td>BRD</td>
<td>LambSoft Magnetic Format</td>
<td><a href="http://www.dcs.shef.ac.uk/~mikem/fileformats/brd.html">www.dcs.shef.ac.uk/~mikem/fileformats/brd.html</a></td>
</tr>
<tr>
<td>C3D</td>
<td>Biomechanics, Animation and Gait Analysis</td>
<td><a href="http://www.c3d.org/c3d_format.hm">www.c3d.org/c3d_format.hm</a></td>
</tr>
<tr>
<td>CSM</td>
<td>3D Studio Max, Character Studio</td>
<td><a href="http://www.dcs.shef.ac.uk/~mikem/fileformats/csm.html">www.dcs.shef.ac.uk/~mikem/fileformats/csm.html</a></td>
</tr>
<tr>
<td>GTR, HTR &amp; TRC</td>
<td>Motion Analysis</td>
<td><a href="http://www.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/">www.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/</a> {HTR.html, TRC.html}</td>
</tr>
</tbody>
</table>

Table 2-1: Common Motion Capture File Formats (from Meredith and Maddock, 2001)

According to Meredith and Maddock (2001), the BVH⁵, HTR⁶ and ASF/AMC⁷ formats tend to be the more common formats used in industry today. These three formats are now evaluated in terms of the pros and cons they provide:

---

⁵ BVH – BioVision Hierarchical Motion File Format.
⁶ HTR – Hierarchical Translation-Rotation Motion File Format.
⁷ ASF/AMC – Acclaim Skeleton File/ Acclaim Motion Capture.
2.3.1 Acclaim Data (ASF/AMC)

Acclaim Entertainment Inc., a gaming company founded in 1987, was amongst the first companies to use motion capture in the entertainment industry. They developed two file formats that are used to store skeletal structure and motion data of a 3D character. The .ASF (Acclaim Skeleton File) file describes the actual character and its hierarchy of joints, while the .AMC (Acclaim Motion File) file describes the corresponding motion data for the character.

One advantage of this file format, according to Lander (1998), is that it is one of the most comprehensive file formats. This allows for any character topology to be implemented. Another advantage of this format is that the range of motion of joints may be limited, making it a very flexible file format. (Unknown A, 2003)

However, Lander (1998) goes on to say that the Acclaim data format, although comprehensive, is also the most complicated out of the three formats. Unknown A (2003) also states that although this format is very common, it has only been implemented correctly in a few CG\(^8\) packages which means that fatal incompatibility problems are frequent. One last disadvantage, according to Unknown A (2003), is that the format is non-encapsulated (i.e. the character structure is separated from the motion data). In other words, because any .AMC motion file corresponds to a single .ASF file, if the wrong skeleton is used, the resulting character movement will look unnatural.

2.3.2 BioVision Data (BVH)

The BVH (BioVision Hierarchical) file format was originally created by BioVision, a motion capture services company, as a way to provide motion capture data to their customers. (Thingvold, 1999) Like the Acclaim data format, the BVH file format permits the description of any character topology. However, unlike the Acclaim data, BVH data encapsulates the skeleton information as well as the associated motion data within a single file.

\(^8\) CG – Computer Graphics.
According to Thingvold (1999), the BVH format is an excellent all around format, its only drawback is the lack of a full definition of the basis pose. BVH files only define translational offsets of children segments from their parent, no rotational offset is defined. Also, the BVH file format lacks explicit information for how to draw the joint segments. (Thingvold, 1999)

However, the BVH file format is still an easy-to-read ASCII format making it relatively popular. Lander (1998) Due to its popularity, the format is supported by many motion capture devices and production houses. Thus obtaining data of this format is relatively easy compared to the other file formats.

2.3.3 Motion Analysis Data (HTR)

The HTR (Hierarchical Translation-Rotation) format was developed as a native skeleton format for the Motion Analysis skeleton generating software. It was created as an alternative to the BVH format because of deficiencies in the BVH specification.

Thingvold (1999) also states that the HTR file format is an excellent all around format: it includes lots of flexibility in terms of data types and ordering, and it has a complete basis pose specification where the starting point for both rotations and translations are given.

The main drawback associated with the HTR format is that there are fewer free HTR sample files available on the Internet compared to the other two formats.

2.4 Summary

The initial sections of this chapter have illustrated the importance and extent of research being done in the areas of motion synthesis and editing. A number of motion editing tools have been created that enable animators to re-use existing motion capture data, by applying the same motion to different characters, by constraining the motion to satisfy user specifications or by blending several motion files together to create novel movements. The focus of my research, as identified by the two foundation papers discussed in the first
section of this chapter, is on the last method of motion synthesis mentioned above: motion synthesis by cutting and pasting segments of motion from various example motion files.

The last section of this chapter identified some motion data file formats in use today and compared three of the most common formats in terms of their advantages and disadvantages. The choice of which file format to use for the implementation of my motion editing tool will be discussed and explained in the next chapter.
3 Design

As already mentioned, the main objective of this thesis is to research and implement some of the techniques used by Lee et al (2002) and Arikan and Forsyth (2002) to generate new motion from existing motion capture data. It is however, important to note that it is not the purpose of this work to create an exact duplicate of what has already been done in this field. With this in mind, it is time to explore some of the design choices that were made and the issues that needed to be considered before implementation and development could begin. This chapter begins by clearly stating the requirements of the system and identifying the major design issues that were addressed. The requirements and interface design of each module forming part of the application are then discussed and finally the design of the data structures used during implementation is discussed.

3.1 System Requirements

Based on the project objectives outlined in Section 1.5, four main requirements of the system can be identified:

- An archive of motion files must be compiled. This archive must consist of motion files that can be compared. In other words, a single motion file format must be chosen and used throughout design and implementation of the system.
- The system must allow for the modelling and animation of 3D articulated skeletons as described by the motion capture data used.
- The system must be able to synthesize new motion by combining segments from different motion files based on some simple user input.
- Lastly, the system must allow for the export of this new motion to Blender, an open source, 3D modelling and animation environment.

3.2 Design Issues

After identifying the above requirements, there is now a need to discuss some of the initial design issues and choices that must be considered, before any further progress can be
made. Firstly, the choice of which motion file format to use is discussed. Next, the reason why it is necessary to build a motion file converter is explained and lastly the choice of which interaction technique to use, is motivated.

3.2.1 Choice of Motion File Format

Since it is imperative that the motion synthesis tool be able compare two motion files from the underlying motion archive, it is necessary to ensure that the raw motion files are of the same format. As shown previously, there are a number of different motion file formats that are in use today. After researching the various advantages and disadvantages of these formats, I decided to use the BVH\textsuperscript{9} file format. The motive behind my choice is three fold:

- Firstly, BVH tends to be one of the more common formats used. As already mentioned, many production houses and motion capture systems support this format. Thus, there are a number of sources on the Internet that together provide a wide variety of free BVH motion sequences.
- Secondly, good format specification documentation is available for the BVH file format, which made it easier to parse and analyse the data stored by each BVH file. More details on the methods employed for parsing the BVH files will be given at a later stage in the chapter.
- Lastly, each BVH file is divided into two sections; the first describing the skeleton hierarchy for the avatar and its initial pose and the second describing the frame by frame motion of the avatar. Using this structure, it is easy to distinguish between the skeleton object and its associated motion data. Details of the data structures that result from parsing and interpreting the BVH files are also outlined later on in the chapter under Section 3.6 entitled “Design of the Data Structures Used”.

\textsuperscript{9}BVH – BioVision Hierarchical Motion File Format.
3.2.2 The Need for a Motion File Conversion Tool

Having chosen a specific motion capture file format, I began building up my motion archive. The implementation issues involved with this process are discussed fully in Chapter 4 “Implementation” in Section 4.2, “Compiling the Motion Data Archive”.

After collecting a substantial amount of raw data in the form of BVH motion files from a number of different sources, I realised it would be necessary to analyse the different skeletal structures\(^\text{10}\) of the BVH files to ensure that it would still be possible to compare them later on. Figure 3-1 illustrates the results obtained when comparing BVH files from eight different sources (shown in Table 3-1 below) based on the number of joints (bones) each describe.

<table>
<thead>
<tr>
<th>Source</th>
<th>Site</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><a href="http://www.centralsource.com">www.centralsource.com</a></td>
<td>dribble.bvh</td>
</tr>
<tr>
<td>2</td>
<td><a href="http://www.charactermotion.com">www.charactermotion.com</a></td>
<td>dude_bthhnd.bvh</td>
</tr>
<tr>
<td>3</td>
<td><a href="http://engr.oregonstate.edu/~zier/Assignment4BVH/">http://engr.oregonstate.edu/~zier/Assignment4BVH/</a></td>
<td>Ron_slow_walking1.bvh</td>
</tr>
<tr>
<td>4</td>
<td>Unknown1</td>
<td>run.bvh</td>
</tr>
<tr>
<td>5</td>
<td>Unknown2</td>
<td>pwalkm10.bvh</td>
</tr>
<tr>
<td>6</td>
<td>Unknown3</td>
<td>sample_move.bvh</td>
</tr>
<tr>
<td>7</td>
<td><a href="http://www.Help3d.com">www.Help3d.com</a></td>
<td>ballet16.bvh</td>
</tr>
<tr>
<td>8</td>
<td><a href="http://www.e-motek.com/entertainment/index.htm">www.e-motek.com/entertainment/index.htm</a></td>
<td>head4.bvh</td>
</tr>
</tbody>
</table>

\(^{10}\) Skeletal structure refers to the character’s make-up in terms of bones and joint segments.

Table 3-1: Source References and File Names of the BVH Files Compared.

![Figure 3-1: BVH File Comparison Based on the Number of Joints Defined in the Skeleton Hierarchy.](chart.png)
Figure 3-1 shows that although most of the sources investigated use twenty three joints to define their skeletons, there are still some other sources that use varying numbers of joints. Also, after further analysis, it was discovered that the names used to describe each joint did not match from one source to another. This meant that it would not be possible to compare motions across sources using the existing BVH files.

Therefore, in order to compare BVH files from multiple sources, it is necessary to define each BVH file as having a common skeleton hierarchical structure. Hence, the need to develop a tool that could convert any BVH file into a BVH file describing the same motion sequence, but using a standard maximal skeleton hierarchy. The chosen maximal skeleton hierarchy is identified and discussed in the next section.

### 3.2.3 Design of the Maximal Skeleton Hierarchy

The “Maximal Skeleton Hierarchy” is the name I’ve given to the chosen subset of joints and their corresponding relationships that represent the most common character structure described by the BVH files I have collected so far. In other words, the maximal skeleton hierarchy represents a multi-jointed structure that contains as many common bones between the various BVH files as possible. During the design of the maximal skeleton hierarchy, it became obvious that due to the range of bone names used to describe the different joints in each BVH file, a list of synonyms of the various bone names would have to be compiled. The following table illustrates the joint synonyms found thus far. Synonyms for the left-hand side of the body are given, but corresponding synonyms for the right-hand side of the body also exist.

<table>
<thead>
<tr>
<th>Maximal Skeleton Joint Name</th>
<th>Synonym</th>
<th>Maximal Skeleton Joint Name</th>
<th>Synonym</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hips</td>
<td>pelvis</td>
<td>LeftHand</td>
<td>leftwrist</td>
</tr>
<tr>
<td>Abdomen</td>
<td>lowerback</td>
<td>LeftUpLeg</td>
<td>lefthip</td>
</tr>
<tr>
<td>LeftUpArm</td>
<td>leftshoulder</td>
<td>LeftUpLeg</td>
<td>lefthigh</td>
</tr>
<tr>
<td>LeftLowArm</td>
<td>leftelbow</td>
<td>LeftLowLeg</td>
<td>lefthshin</td>
</tr>
<tr>
<td>LeftLowArm</td>
<td>leftforearm</td>
<td>LeftLowLeg</td>
<td>leftknee</td>
</tr>
</tbody>
</table>

*Table 3-2: Synonyms of Maximal Skeleton Joint Names*
Before **BVH** motion files from different sources can be compared to create new motion, they need to be converted to represent the maximal skeleton hierarchy. Implementation details concerned with the development of the motion conversion tool are discussed in the next chapter. For now, the chosen maximum skeleton hierarchy is shown below: Figure 3-2 shows the relationships between the various joints making up the skeleton hierarchy while Figure 3-3 shows a graphical representation of the maximal skeleton hierarchy.

![Maximal Skeleton Hierarchy](image)

**Figure 3-2: Maximal Skeleton Hierarchy**

It is important to note that most of the researchers, who have implemented similar approaches to the one being developed here, have compiled their underlying motion archive from raw data obtained from a single source. This implies that there was no need for them to build a conversion tool as all their motion files described the same skeleton hierarchy.
3.2.4 Choice of the User Interaction Technique

As explained in Chapter 2, the tool developed by Lee et al (2002) provides three different user interaction techniques that can be used to control the avatar: choice, sketching and vision. Due to the time constraints of this project, it would not be practical to implement all three techniques. Thus, there is the need to make a choice about which control technique to implement. The vision option, which allows the user to act out a desired motion in front of a video camera, is not feasible for me to implement as the required equipment is not available to use. Both the sketching and choice techniques are viable options, however, in the end I chose to implement the choice technique mainly because it most closely relates to the underlying data structures, making it the easiest technique to put into practice. Essentially, this interaction technique allows the user to choose which transitions occur during the process of motion synthesis.
3.3 System Architecture

A simple system overview or architecture can be developed by analysing the system requirements that were identified in Section 3.1. This architecture is illustrated in Figure 3-4 below.

![Figure 3-4: System Architecture illustrating individual System Components](image)

As can be seen from the diagram above, the system is made up of four components or modules, outlined below:

- **The Motion Archive**: This component represents the collection of raw data that will be used to evaluate the conversion and synthesis tools. The motion archive forms a pool of BVH files, originating from a number of different sources.
- **The Motion File Conversion Tool**: This module takes a BVH file, from the underlying motion archive, as input and converts it into a BVH file that has the same motion but applied to the maximal skeleton hierarchy. As already explained, this step is critical if BVH files from multiple sources need to be compared.
• The **Motion Display and Synthesis Tool**: This component of the system deals with rendering the animations described by the converted **BVH** files as well as providing an interface that allows the user to synthesize novel motion by combining several motion segments from different **BVH** files.

• The **Blender Export Component**: The last component allows the edited motion sequence to be saved and exported to Blender, where it can be modified further.

Having suggested a simple architecture that identifies the basic relationships that exist between the various components that make up the system, it is now possible to discuss the two main components (Motion File Converter and the Motion Display and Synthesis Tool) in greater detail. This is done with particular emphasis on the individual requirements of each module and the user interface design of each component.

### 3.4 Motion File Converter Design

The following section begins by outlining the initial requirements of the motion file converter that were identified during the design phase. A simple interface design is then proposed and any design considerations relating to user interaction are discussed.

#### 3.4.1 Module Requirements

The following list consists of the main requirements of the motion file converter:

- The conversion tool must be able to read in any valid **BVH** file.
- It must be able to extract the relevant data relating to the original character skeleton and its motion.
- It must be able to lookup joint names in a list of synonyms (similar to the one described in Table 3-2) and replace them so that they refer to valid maximal skeleton joint names.
- It must be able to convert the data relating to the original character skeleton so that the same motion is performed by the maximal skeleton.
It must allow the user to save the converted BVH file, so that it can be used at a later stage by the display and synthesis tool.

### 3.4.2 Interface Design

By considering the above requirements, there are a number of interface design issues that can be addressed immediately. First of all, the user must be able to select which BVH file they would like to convert. This can be done easily enough using a standard open file dialog component. Secondly, it would be a good idea to display some feedback to the user based on the results of the conversion process. The original character motion and the converted character motion should be displayed as feedback to the user. Also, a number of camera positioning controls and playback options should be provided to make it easier for the user to view the converted motion. The proposed user interface is illustrated below:

Figure 3-5: Proposed User Interface for Motion File Converter

### 3.5 Motion Display and Synthesis Tool Design

The main requirements of the motion display and synthesis tool are discussed next. The proposed interface design for the tool is also established and some of the design considerations involved with user interaction are acknowledged.
3.5.1 Module Requirements

The following list consists of the main requirements of the motion display and synthesis tool:

- The display tool must be able to parse a valid BVH file and render the character and its movement on screen.
- Camera control and playback control buttons should allow the user to easily target the segment of motion that they are interested in.
- On the motion synthesis side, the tool should process a collection of BVH files, selected by the user and describing independent motions of the same character, by identifying the best possible transitions between the motions.
- The motion synthesis tool should provide an intuitive way for the user to choose which transitions are desired in the new motion.
- The tool should also allow the user to prune the synthesized motion before saving it to file.
- Finally, the tool must provide an option for exporting the newly created BVH file to Blender.

3.5.2 Interface Design

According to Lee et al (2002), one of the most challenging areas of avatar animation and control is providing a natural interfacing technique that allows the user to choose from a set of underlying character behaviours usually with a low-dimensional input device. Input from devices such as mice and joysticks typically indicates a position, velocity or behaviour. In this case, use of the mouse with the proposed interface caters for the last option. As already discussed in Section 3.2.4, the “choice” interfacing technique developed by Lee et al (2002) is the main interfacing technique implemented here. A simple design of the initial user interface for the motion display and synthesis tool is shown in Figure 3-6 next. The proposed menu options are shown in Figure 3-7 afterwards.
3.6 Design of the Data Structures Used

This section describes the design of the major data structures that are used by the system to store the information extracted from the BVH files. In order to understand the reasoning behind the composition of these data structures, it is important to briefly examine some theory relating to the constitution of a BVH file. To begin with, a number of standard terms are defined.
3.6.1 Terminology

The following list outlines some of the more important keywords that will be used to identify and describe the different components that form a BVH file:

- **Skeleton** – the whole character that the motion capture data applies to. (Lander, 1998)
- **Bone** – the basic entity in representing a skeleton. “Each bone represents the smallest segment within the motion that is subject to individual translation and orientation changes during the animation.” (Meredith and Maddock, 2001)

Therefore, a skeleton is comprised of a number of bones usually in a hierarchical structure as shown previously in Figure 3-2 and Figure 3-3. In this context, the term **joint** is often used instead of bone.

- **Channel/Degree of Freedom (DOF)** – a parameter that describes the position, orientation or scale of a bone. The changes in channel data over time give rise to the animation. (Meredith and Maddock, 2001)
- **Animation Stream** – a combination of channels.
- **Frame** – refers to a particular slice of time where the channel data for each bone is defined. (Lander, 1998) Every animation is comprised of a number of frames, where the normal frame rate is about 30 frames per second.

3.6.2 A Sample BVH File

Listing 3-1 shows part of a sample BVH file. The main components of the file format are then discussed afterwards.
As already mentioned, the BVH file format has the advantage of storing both the skeleton hierarchy and the motion data in the same file.

As seen in the BVH file fragment in Listing 3-1, the definition of the skeleton hierarchy begins with the word “HIERARCHY”. The very next line starts with “ROOT” which is
then followed by the name of the root segment or joint for the hierarchy that is to be defined. This is the parent bone for all other bones in the hierarchy. In the example given, the joint *Hips* is the root of the defined skeleton. The remaining structure of the skeleton is defined in a recursive nature where each joint’s definition, including any children, is encapsulated in curly braces, which is delimited on the previous line with the keyword “**JOINT**” followed by the name of the joint. (Meredith and Maddock, 2001) The first piece of information within a joint’s definition is the offset of that segment from its parent. This is identified by the keyword “**OFFSET**” and essentially describes the length of the bone between the current joint and its parent. The line following the offset contains the channel information. This has the "**CHANNELS**" keyword followed by a number indicating the number of channels and then a list of that many labels indicating the type of each channel. The most common setup for a *BVH* file is for the root joint to have 6 channels, 3 for positional data and 3 for rotational data, while the rest of the joints only have rotational data. Each branch in the hierarchy ends with a joint named “**End Site**”. The end site information ends the recursion and indicates that the current segment is an end effector (has no children). This joint is used to determine the length of the last bone (Lander, 1998 and Thingvold, 1999).

Following the hierarchy section is the motion section, which begins with the keyword “**MOTION**”. This section describes the animation of each bone over time. In other words, it stores channel data for each bone in the order they were seen in the hierarchy definition, where each line of float values represents an animation frame. The number of frames the motion file describes is given after the keyword “**Frames:**” and the sample rate of the motion data is defined after the keywords “**Frame Time:**” (Lander, 1998 and Thingvold, 1999).

### 3.6.3 Parsing the BVH File

As mentioned previously, the skeleton hierarchy of a *BVH* file is essentially recursive in nature, which means that a simple recursive decent parser can be used to extract the skeleton data. The motion data, on the other hand, can be parsed line by line and stored in a matrix of channels, where each channel describes three separate joint angles, representing...
rotation or translation about the X, Y and Z axes. The main data structures used to store the data are described next.

3.6.4 Skeleton Hierarchical Data

Two main data structures are used to describe the hierarchy and initial pose of the skeleton, namely the Skeleton Node class and the Skeleton class. Skeleton Node objects are used to store information relating to each joint (bone) in the skeleton. Two important points to note are that each Skeleton Node, except the root node, has only one parent and that each Skeleton Node stores a list of its children nodes. A Skeleton object represents the hierarchical skeletal structure as a simple tree structure formed from a linked list of Skeleton Nodes. Each branch of the tree is terminated with a Skeleton Node named “End” + [Parent’s Name]. The most important attributes and operations of each of these classes are shown in the listings below:

<table>
<thead>
<tr>
<th>Skeleton Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>- name : String</td>
</tr>
<tr>
<td>- offset : Vector</td>
</tr>
<tr>
<td>- numChildren : int</td>
</tr>
<tr>
<td>- children : SkeletonNode**</td>
</tr>
<tr>
<td>- parent : SkeletonNode*</td>
</tr>
<tr>
<td>+ addChild()</td>
</tr>
</tbody>
</table>

Listing 3-2: Main Attributes and Operations of the SkeletonNode Class

<table>
<thead>
<tr>
<th>Skeleton</th>
</tr>
</thead>
<tbody>
<tr>
<td>- root : SkeletonNode*</td>
</tr>
<tr>
<td>- mov : Movements*</td>
</tr>
<tr>
<td>- frame : int</td>
</tr>
<tr>
<td>+ findSkeletonNode()</td>
</tr>
<tr>
<td>+ addBone()</td>
</tr>
<tr>
<td>+ setFrame()</td>
</tr>
<tr>
<td>+ setMovements()</td>
</tr>
<tr>
<td>+ renderSkeleton()</td>
</tr>
<tr>
<td>+ writeBVH()</td>
</tr>
</tbody>
</table>

Listing 3-3: Main Attributes and Operations of the Skeleton Class

37
A Maximal Skeleton class has also been designed which inherits attributes and operations from the Skeleton class, but sets up a skeleton hierarchy that represents the maximal skeleton defined in Section 3.2.3.

3.6.5 Motion Data

The motion data extracted from the **BVH** file can be stored using a matrix of channels as mentioned above. Essentially, each row in the matrix represents the motion data for all joints in the skeleton for a single frame. Each column stores a single channel object. The attributes and operations of the channel and movements classes are shown below:

<table>
<thead>
<tr>
<th>Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>-rot : Quaternion</td>
</tr>
<tr>
<td>-trans : Vector</td>
</tr>
<tr>
<td>-Euler : Vector</td>
</tr>
</tbody>
</table>

**Listing 3-4:** Main Attributes and Operations of the Channel Class

<table>
<thead>
<tr>
<th>Movements</th>
</tr>
</thead>
<tbody>
<tr>
<td>-filename : String</td>
</tr>
<tr>
<td>-frame : int</td>
</tr>
<tr>
<td>-names : ChannelNames</td>
</tr>
<tr>
<td>-value : Channel*</td>
</tr>
<tr>
<td>-frameinterval : double</td>
</tr>
<tr>
<td>+getChannelEntry()</td>
</tr>
<tr>
<td>+parseBVH()</td>
</tr>
</tbody>
</table>

**Listing 3-5:** Main Attributes and Operations of the Movements Class

The “value” attribute of the Movements class is an array of Channel objects that represents the matrix of joint angles extracted from the **BVH** file. A high level view of this array is shown in Table 3-3:
### Table 3-3: A 2D array of Frames and Channels representing a Motion segment of a BVH file

<table>
<thead>
<tr>
<th>Frame 1</th>
<th>Hips Translation</th>
<th>Hips Rotation</th>
<th>Chest Rotation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>((X_{\text{trans}1}, Y_{\text{trans}1}, Z_{\text{trans}1}))</td>
<td>((X_{\text{rot}1}, Y_{\text{rot}1}, Z_{\text{rot}1}))</td>
<td>((X_{\text{rot}1}, Y_{\text{rot}1}, Z_{\text{rot}1}))</td>
</tr>
<tr>
<td>Frame 2</td>
<td>((X_{\text{trans}2}, Y_{\text{trans}2}, Z_{\text{trans}2}))</td>
<td>((X_{\text{rot}2}, Y_{\text{rot}2}, Z_{\text{rot}2}))</td>
<td>((X_{\text{rot}2}, Y_{\text{rot}2}, Z_{\text{rot}2}))</td>
</tr>
</tbody>
</table>

#### 3.6.6 Unified Modelling Language (UML) Description of the System

The proposed design of the system can be described using two UML diagrams. The first, shown in Figure 3-8 below, illustrates the relationships that exist between the different data structures described above. The second UML diagram, shown in Figure 3-9, illustrates the relationships between the main programs and the user interface classes. There are four main user interface classes. The SceneViewer class is the main interface class that is made up of a class used to render various scene objects (SceneViewerGLWorld), a class used to control the animations (SceneViewerControlWidget) and a class used to represent the menu bar (SceneViewerMenuBar).

![Figure 3-8: UML Description of the Problem Domain Classes](image-url)
3.7 Summary

This chapter began by stating the requirements of the system, followed by a discussion of several important design decisions that had to be made before implementation could begin. Firstly, it was decided that the BVH file format should be used throughout implementation because of its intuitive structure and its availability on the Internet. It was discovered that there was need for a motion file converter so that BVH files from different sources could be compared during the Markov process. Also, it was decided that the user interaction technique to be implemented would be the “choice” technique developed by Lee et al (2002). Then the requirements and interface design of the Motion File Conversion Tool and the Motion Display and Synthesis Tool were discussed and finally the proposed data structure designs were discussed and UML descriptions of the problem domain components and the graphical user interface components were given.
4 Implementation

The aim of this chapter is to discuss the implementation of the system design introduced in the previous chapter. The chapter begins by discussing the different components of the implementation environment and motivation for the choice of implementation languages and platforms is given. Thereafter, implementation issues relating to each sub-component of the system are examined and discussed. Where appropriate, a distinction is made between the approaches taken by Lee et al (2002) and Arikan and Forsyth (2002) and the approach I have adopted.

4.1 Implementation Languages and Platforms

4.1.1 Linux

Although the tools required for implementing this project are available through both Windows and Linux, Linux RedHat 8.0 was the chosen implementation platform. The reason for this is two fold:

- First and foremost, the tool that I develop must be compatible with the other graphics and virtual reality related tools already being used within the Computer Science Department. Since most of these other tools are Linux based programs, it is important for ease of integration to carry out any implementation using the Linux platform.
- Secondly, I have experience using the Linux platform for developing graphics based applications due to the Graphics Course taken at the beginning of the year.

4.1.2 OpenGL

In order to quickly develop an interactive 2D or 3D graphical application, it is often necessary to make use of a graphics interface or library so that direct manipulation of the graphics hardware is avoided. I have decided to make use of the OpenGL graphics
interface for several reasons. Firstly, I have had some previous experience with OpenGL and secondly, OpenGL is cross-platform and is the graphics industry standard for 3D rendering and 3D hardware acceleration. According to the official OpenGL website (Unknown B, 2003), OpenGL is the industry’s must widely used, supported and best documented graphics API\(^\text{11}\). Therefore, it is easy to obtain information on implementing OpenGL in hardware and software.

To achieve OpenGL’s hardware independence, commands for windowing tasks as well as commands for obtaining user input were excluded. This seems like it would be a serious drawback to using OpenGL, but it is possible to combine OpenGL with other flexible programming libraries that will handle windowing tasks and obtain user input. (Sepulveda, 1998) The choice of windowing toolkit is discussed next.

### 4.1.3 Qt GUI Application Development Framework

There are a number of windowing toolkits available that complement the OpenGL library. The better known two are GLUT (OpenGL Utility Toolkit) and Trolltech’s Qt windowing toolkit.

According to Sepulveda (1998), GLUT is a programming interface with ANSI C and FORTRAN bindings for writing window system independent OpenGL programs. Although GLUT is relatively easy to learn, it is also fairly small and so does not contain all the features that a sophisticated user interface may require. Due to this fact, it was decided that the Trolltech’s Qt Library would be a better option for implementation.

According to Trolltech’s website (Unknown C, 2003), Qt is the emerging standard for multiplatform C++ GUI application development. Qt allows the programmer to write a single source-tree and then port it to multiple platforms with a simple recompile. One of the main reasons why Qt is so useful for 3D graphics programming is because it contains an extension module for OpenGL which makes it easy to develop and maintain a fully-functional user interface for an OpenGL program.

\(^{11}\text{API (Application Programming Interface)}\)
4.1.4 C++

The choice of C++ as the base implementation language hinges on the fact that the Qt interface is geared towards C++ application development. Also, C++ was the language used during the Graphics course that I took earlier on in the year and so I have had experience developing applications with C++, OpenGL and the Qt libraries.

4.1.5 Blender 3D Modelling and Animation Environment

“Blender is the first and only fully integrated 3D graphics creation suite allowing modelling, animation, rendering, post-production, real-time interactive 3D and game creation and playback with cross-platform compatibility – all in one tidy, free downloadable package.” (Unknown D, 2003) Blender also allows users to explicitly manipulate 3D objects and their attributes using a scripting language known as Python. According to Strubel (2000), Python is popular on every platform and is very easy to learn. Thus the reason for choosing Blender as the external modelling environment was mainly based on the fact that Blender is free and there is a lot of documentation and help available on how to write Blender scripts.

4.1.6 Integration of Qt, C++, OpenGL and Blender

As shown by the UML diagrams from the previous chapter, the problem domain components have, where possible, been separated from the user interface components so as to improve the readability and maintainability of the code. Therefore, a combination of Qt library code and C++ functions has been used to provide the link between the interface components and the rest of the system. The Qt library supports a signal to slot framework which allows for the handling of communication between objects during runtime. (Palmer, 2001) Every time a user initiated event occurs, a signal is created and sent to the connected slot. The slot then sets flags or calls functions that affect the problem domain components and the rest of the system.
The results of the user’s actions are then rendered using OpenGL commands and displayed on screen using the widget classes provided by the Qt libraries. In order to export the edited motion sequences to Blender, a script needs to be written in Python to read in a **BVH** file and automatically generate the motion curves representing the animation.

The following sections elaborate on the implementation of each of the main components that form the system.

### 4.2 Compiling the Motion Data Archive

Having chosen a specific motion file format, the next step in implementation was to collect a whole lot of **BVH** files to build up my motion archive. According to Lee et al (2002), the **size** and **quality** of the underlying archive of motion capture data is key to the success of their approach. The archive must be **large** enough so that the likelihood of finding good transitions between different motion sequences is high. While their (Lee et al, 2002) raw data was captured using a Vicon optical motion capture system, such equipment was not available for me to use, so it was necessary to turn to other possible sources of motion capture data. Fortunately, there are several websites that offer free motion capture data for anyone to download. References to these websites are provided on my “Resources” page, accessible from my project webpage as well as from Appendix C at the end of the thesis. In fact, it is this growing availability of significant quantities of free motion data that has led to a great deal of research focusing on methods for modifying and varying existing data (Lee et al, 2002).

An important point to note at this stage is the difference in **quality** between the motion archive used by Lee et al (2002) and the motion archive compiled for this project. Firstly, the archives used by both Lee et al (2002) and Arikan and Forsyth (2002) are based on only one subject’s motion, while the archive compiled for this research is built up from several sources which implies the use of different skeleton structures. As already mentioned in section 3.2.2, this fact then led to the need for a motion file converter. Another difference that can be identified is that the motion sequences I have used to build up the archive vary a great deal in terms of their content. This can be shown by the wide range of headings used on my resources webpage, to categorize the motions in the archive.
The motion archive built by Lee et al (2002) comprises of several motion sequences all with the same theme. This improves the chances of finding a decent match between different motion segments.

In order to help with the process of updating the online version of the motion archive, a simple shell script was developed that automatically adds all recently collected motion files to the archive’s web interface. Essentially, the script traverses through the motion file category folders stored on my computer (see Listing 4-1 below), creating a new HTML page for each category it comes across. Also, it compresses all the motion files in each category and then provides download links for the zipped files and for each individual file as well (see Listing 4-2 below).

```
cd ResearchProject/Assets/Resources/BVHSampleFiles
for dir in $(ls)
    do
        echo going into dir: $dir
        echo "\n\n<h2 align="center">\n<font face="Comic Sans MS">\n<a href="$dir.html">$dir</a>\n</font></h2>" >>$restdestfile
    done
```

**Listing 4-1:** Shell Script Snippet showing the creation of the Motion Category Links

```
#Loop for each BVH file in Source Directory
for file in $(ls *.bvh)
    do
        echo "\n<tr>" >>$destfile
        echo "\n<tr>\n<td><a href="Assets/Resources/BVHSampleFiles/$dir/site/$file">$file</a></td>" >>$destfile
        echo "<td>&nbsp;</td>" >>$destfile
        echo "<td>\n<font size="4">BVH</font></td>" >>$destfile
        echo "</tr>" >>$destfile
    done
    echo "</table>" >>$destfile
    echo "</br>" >>$destfile
```

**Listing 4-2:** Shell Script Snippet showing BVH Table Generation

The entire shell script code can be found in Appendix A near the end of the thesis.
4.3 Building the Motion File Converter

This section discusses the main implementation issues involved with developing the motion file converter. The conversion process consists of two main phases. The first replaces the original joint names with those used by the maximal skeleton and the second phase establishes the new skeleton’s initial pose and adapts the motion values of the converted BVH file accordingly. Two alternative methods have been implemented to solve the problem introduced in phase two. The first method requires that the rotation values for each joint in the motion data be modified. However, at present the rotation data is stored as Euler angles which are not easily operated on directly using standard mathematical formulae. For this reason, we begin this section by discussing the alternative to Euler angle representation and then continue by describing the two phases that make up the conversion process.

4.3.1 Quaternions vs. Euler Angles

According to Gain (2001), a general rotation can be constructed as a sequence of rotations about 3 mutually orthogonal axes, forming a set of Euler Angles. The joint angles that are used to represent the orientation of bones within a BVH file are stored as a sequence of Euler Angles corresponding to rotations about the X, Y and Z axes. Although historically popular as a representation for rotations, Euler Angles are somewhat flawed and difficult to use when it comes to motion editing techniques such as interpolation, amplification and motion blending. (Gain, 2001) One of the main flaws with the Euler Angle representation of rotations is that the rotations are not unique; there may be many ways of expressing the same rotation. Also, the rotations are not smooth: linearly interpolating between two rotations will not give a smooth path and can often cause gimbal lock\textsuperscript{12}. Thus, performing operations directly on Euler angles can lead to some strange results and should therefore be avoided where possible. An alternative representation of orientation that tends to be easier to use for motion editing operations is the quaternion.

\textsuperscript{12} Gimbal Lock is the phenomenon of two rotational axes of an object pointing in the same direction. Simply put, it means the object won’t rotate the way you think it should. (Unknown E, 2003)
Quaternions represent an orientation by a counter clockwise rotation angle about an arbitrary vector. “Quaternions extend the concept of rotation in three dimensions to rotation in four dimensions. This avoids the problem of gimbal lock and allows for the implementation of smooth and continuous rotation.” (Massimino, 2003) Originally, quaternions were devised as a four-dimensional extension to complex numbers, but another popular notation is the 4D vector notation shown below:

\[ [w, v] \text{ (where } v = (x, y, z) \text{ is called a "vector" and } w \text{ is called a "scalar")} \]

Using this notation, a number of standard, basic quaternion operations have been defined that help the animator implement the motion editing techniques mentioned earlier. (See Table 4-1 below)

<table>
<thead>
<tr>
<th>Table 4.1. Basic operations using quaternions.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Addition:</strong> ( q + q' = [w + w', v + v'] )</td>
</tr>
<tr>
<td><strong>Multiplication:</strong> ( qq' = [ww' - v \cdot v', v \times v' + wv' + w'v] ) (( \cdot ) is vector dot product and ( \times ) is vector cross product); Note: ( qq' = q'q )</td>
</tr>
<tr>
<td><strong>Conjugate:</strong> ( q^* = [w, -v] )</td>
</tr>
<tr>
<td><strong>Norm:</strong> ( N(q) = w^2 + x^2 + y^2 + z^2 )</td>
</tr>
<tr>
<td><strong>Inverse:</strong> ( q^{-1} = q^* / N(q) )</td>
</tr>
<tr>
<td><strong>Unit Quaternion:</strong> ( q ) is a unit quaternion if ( N(q) = 1 ) and then ( q^{-1} = q^* )</td>
</tr>
<tr>
<td><strong>Identity:</strong> ( [1, (0, 0, 0)] ) (when involving multiplication) and ( [0, (0, 0, 0)] ) (when involving addition)</td>
</tr>
</tbody>
</table>

Table 4-1: Basic Operations Using Quaternions (from Bobick, 1998)

Therefore, having discussed the advantages of quaternions over Euler Angles, it makes sense to make use of quaternions for the motion editing that occurs in phase two of the conversion process. However, the converted motion data has to be saved as a BVH file again, which means that the new quaternion values will have to be converted back to the corresponding Euler Angles. Essentially, the Euler Angles from the original motion data are converted to quaternions which are then used in the intermediate calculations. The resulting quaternions are then converted back to Euler angles at the end. The implementation issues that result will be discussed later in this section.
4.3.2 Phase 1: Renaming the Joints

For each joint name used by the existing BVH file, a list of valid synonyms is searched and if the name is found, it is changed to the corresponding maximal skeleton joint name. This eliminates the problem caused by different naming conventions between sources of BVH files. A map data structure called “OtherNames” is used to store the joint name synonyms as shown below:

```
std::map<std::string, std::string> OtherNames;
```

**Listing 4-3: Setting up a map to store the Joint Synonyms**

```
// Create Lookup table for other joint and bone names
OtherNames["hips"] = "Hips";
OtherNames["hip"] = "Hips";
OtherNames["pelvis"] = "Hips";
```

**Listing 4-4: Code Snippet showing part of the map Initialisation**

Each time a new joint is parsed in the original motion file, its name is looked up in the map data structure and the corresponding maximal skeleton name is returned.

```
std::string test;
test = activejoint;
test = skel.lowercase(test);

std::string newname = skel.findRealName(test);
if (newname =="
 {
    std::cerr << "No entry for " << activejoint << "\n";
}
else
 {
    std::cout << "Found " << activejoint << "\n";
    activejoint = newname;
}

sn = skel.addBone (activejoint, parent);
```

**Listing 4-5: Code Snippet showing how the Skeleton Joints are renamed.**
4.3.3 Phase 2: Establishing the Initial Skeleton Pose

The second phase to the motion file conversion process involves setting up the actual maximal skeleton hierarchy. This involves establishing the bone lengths and the initial rotation values of each joint which together form the maximal skeleton’s initial pose. An important point to note is that as the original motion file is parsed, any joints that exist in the maximal skeleton hierarchy, but are not described by the original file are identified and their joint angles are set to zero. Therefore, their rotation will only be determined by their parent’s motion and their grand-parents motion and so on and so on. Two different methods have been implemented to perform the above described tasks:

**Method 1: Modifying the Joint Angles to Match the Initial Skeleton Pose**

The first method involves setting up the maximal skeleton’s initial pose based on a standard calibration stance such as the one shown in Figure 3-3 earlier. Thus, the initial pose of the original motion file is totally ignored. However, the joint angles describing the motion of the skeleton are all calculated relative to the skeleton’s initial pose. This means that when converting **BVH** files with different initial poses to that of the maximal skeleton, it is necessary to slightly offset each joint angle so that the new initial pose is taken into account and the motion sequence is not corrupted.

The process of updating the motion values to represent the same motion using the maximal skeleton’s initial pose can be defined recursively. This is where the quaternion operations come in handy. The following section of code illustrates the recursive method that iterates through each joint in the skeleton and calculates the change in orientation values required by each joint in the maximal skeleton hierarchy in order to produce the converted movements.
Listing 4-6: Recursive method to calculate the change needed in Orientation Values for each joint

Quaternion \( R \) in the previous method represents the change in orientation required for the current joint. Each SkeletonNode Object stores this value in the variable called \texttt{changeOri}. Once each joint has a corresponding value assigned to \texttt{changeOri}, its new joint angles are calculated using quaternion multiplication. For each joint in the skeleton hierarchy and each frame in the motion, the original joint angles are multiplied by the \texttt{changeOri} quaternion to produce new joint angles which represent the motion of the maximal skeleton.

\textbf{Method 2:} Modifying the Bone Lengths of the Original Skeleton Structure

With the second method, the joint angles from the original motion data are not altered at all. Therefore, there is no need to use quaternions in the conversion process. Instead of using a standard maximal skeleton pose for every converted motion file, the original motion file’s initial pose is used, but the hierarchy and bone lengths are updated to represent the maximal skeleton structure. The code shown in Listing 4-7 outlines how the
bone lengths are updated to the size of the maximal skeleton bones, while keeping the original bone orientations.

```cpp
SkeletonNode * pn = dstjoint->parent;
if (pn == NULL)
{
    std::cerr << "Remapped Hips in the source skeleton\n";
    // update bone position.
    Point origin (0.0, 0.0, 0.0);
    dstjoint->position = origin + dstjoint->offset;
}
else
{
    bonelength = (dstjoint->offset).length();
    direction = srcjoint->offset;
    direction.normalize();
    dstjoint->offset = bonelength * direction;
    // update bone position.
    dstjoint->position = pn->position + dstjoint->offset;
}
```

**Listing 4-7: Updating Bone Lengths**

4.4 Building the Motion Display Component

Having completed the first project objective, building a suitable archive of raw motion capture data in a common format, the next step was to develop a simple tool to animate the motion data. As mentioned earlier, implementation of this tool was carried out in C++, making use of the OpenGL graphics library and Trolltech’s Qt windowing toolkit. Animating the motion data found in the BVH files is done in two steps. Firstly, the BVH file is parsed and all the important information is extracted and stored in appropriate data structures. Secondly, the data is processed to produce the avatar’s motion. These steps are described in more detail below:

4.4.1 Parsing the BVH File and Extracting the Relevant Data

A general overview of the parsing process and related data structures has already been discussed in Section 3.6, “Design of the Data Structures Used”. What is left to be discussed is how the parser was implemented. Parsing a BVH file takes place in two stages. First of all, a simple recursive decent parser is used to extract the data relating to
the Skeleton Hierarchy section of the BVH file. Then the Motion section of the BVH file is scanned line by line and the joint angles extracted and stored with their corresponding channels. During the parsing of the channel data, an assumption is made that the ordering of the channels for each joint is in the order Zrotation, Xrotation, Yrotation as this is the order used by most BVH files. At the end of the parsing process, assuming it was successful, there should be a Skeleton object and associated SkeletonNode Objects representing the defined Skeleton Hierarchy. There should also be a Movements Object containing an array of Channels that represent the character’s motion.

4.4.2 Processing the Extracted Data

Before the individual joints of the skeleton can be rendered to the 3D virtual environment, their positions need to be calculated for each frame. The following algorithm is used to obtain the position of a bone segment for a particular frame in the motion sequence:

- Obtain local translation and rotation information.
  - For each joint (Skeleton Node), translation information comes from the offset defined in the hierarchy section of the BVH file.
  - For each joint, rotation information comes from the motion data and the quaternions for each rotation channel.
  - For the root joint, translation information = offset data + root’s Translation channel data stored by “trans” (see Listing 3-4).
- Once the local transformation is created then concentrate it with the local transformation of its parent, then its grand parent etc.

The above described algorithm is implemented using a recursive method, making use of OpenGL’s matrix and stack operations to keep track of parent and child transformations. Once the position of each bone segment is known, all that is left to do is to render a 3D representation of the bone. AC3D, a 3D object editor, was used to create individual 3D OFF objects for each bone by dividing up the skeleton object shown in Figure 4-1 over the page. Parsing and rendering of the OFF object files was done using code developed by Bangay (2003).

---

13 OFF – Object File Format
4.4.3 Camera and Playback Controls

One of the requirements of the motion display tool was to provide camera and playback controls that would allow the user greater flexibility when manipulating the view of the rendered skeleton. The status of a number of flags is used to keep track of which buttons the user has pressed. These flags are constantly checked by the main program and any changes to their status are sent from the control widget to the rendering widget. In this way, the user can play and stop the motion at any time and can even control playback of the animation frame by frame.
The user is also able to change the position of the camera by dragging the mouse pointer on the rendering widget. This allows for translation, rotation and scale operations on whatever scene objects are currently being displayed by the rendering widget. Additional camera controls allow the user to reset the camera position to point back to the origin, to instruct the camera to move with the skeleton or to instruct the camera to remain stationary, but still track the skeleton’s movement. Getting the camera to follow the skeleton is implemented by setting the position of the camera according to the translation of the root joint of the skeleton (usually the Hips). The last camera control mentioned above is implemented in OpenGL by using the gluLookAt() command which specifies the camera position and the centre of its focus (in this case, the root joint of the skeleton, as well).

4.5 Building the Motion Synthesis Component

The next phase in project implementation involved researching and incorporating the motion synthesis algorithms described by Lee et al (2002) so that a preprocessed motion database could be constructed from some of the raw data in the underlying motion archive. This preprocessed motion database could then be used to synthesize new avatar motion based on some simple rules and user input. This section discusses the approach taken to synthesize new motion sequences and how it differs to the one described by Lee et al (2002).

4.5.1 Theoretical Underpinnings

The basic theory behind the approach used by Lee et al (2002) has already been discussed in Section 2.1.2 of this thesis. To recap, they suggest creating a two-layer structure to represent the data in the preprocessed motion database. The higher layer is formed by clustering the data based on similarities among character states while the lower layer represents the results of applying a Markov process to the data. The process of clustering will not be discussed further as it has not been implemented in the chosen approach. However, implementation of the Markov process is at the heart of my project so will be examined in more detail.
4.5.2 Implementation of the Markov Process

Based on similar work done by Schodl et al (2000), the motion data is modelled using a first-order Markov process. The result of the process is represented as a matrix of probabilities \( P_{ij} \) describing the likelihood of transitioning from frame \( i \) to frame \( j \). \( P_{ij} \) is estimated using an exponential function that maps a measure of similarity between frames (\( D_{ij} \)) to a value between 0 and 1 as shown by Equation 1 below:

\[
P_{ij} \propto \exp\left(-\frac{D_{i,j-1}}{\sigma}\right),
\]

Equation 4-1: Maps the distance between frame \( i \) and frame \( j \) to a probability. (Lee et al, 2002)

The constant \( \sigma \) in the above equation is used to control the steepness of the slope described by the exponential function. The following figure illustrates the different graphs obtained by using three different values for \( \sigma \).

![Figure 4-2: Graphs plotting Equation 1 based on 3 different values for \( \sigma \): 1, 3 and 5.](image)

The distance measure \( D_{ij} \) takes into account the weighted differences of joint angles as well as the weighted differences of joint velocities. Analysing the joint angles ensures that the difference in character pose from one frame to another is not significant, while analysing the joint velocities helps to preserve the dynamics of the motion. The equation used to
calculate $D_{ij}$ is shown below. The first term represents the weighted differences of joint angles, while the second term represents the weighted differences of joint velocities. Parameter $v$ weights the velocity differences with respect to position differences. (Lee et al, 2002)

\[
D_{ij} = d(p_i, p_j) + \nu d(v_i, v_j).
\]

**Equation 4-2:** Distance Function used to measure Similarity between frames (from Lee et al, 2002)

Lee et al (2002) use Euclidean differences to calculate the second term and use the following equation to calculate the first term:

\[
d(p_i, p_j) = \|p_i,0 - p_j,0\|^2 + \sum_{k=1}^{m} w_k \| \log (q_{j,k}^{-1} q_{i,k}) \|^2
\]

**Equation 4-3:** Position Differences Function (from Lee et al, 2002)

Where $p_{i,0}$ is the translational position of the character at frame $i$, $q_{i,k}$ is the orientation of joint $k$ with respect to its parent in frame $i$, and joint angle differences are summed over $m$ rotational joints. Weights represented by $w_k$ are used to identify important joints in the skeleton structure. Weights are set to one for significant joints and zero for joints which have less impact on visible differences between poses. (Lee et al, 2002)

Now that the approach used by Lee et al (2002) has been outlined, it is necessary to explore how the Markov process has been implemented in this project. To begin with, the parameter $\sigma$ has been set to 3 and the parameter $v$ has been set to 2. These values were discovered merely through experimentation and a process of trial and error. When calculating the velocity differences (second term of $D_{ij}$), only the root joint velocities are taken into account. This is mainly due to the fact that checking too many joints tends to slow down the whole process tremendously. The code listing over the page shows the method that is used to calculate $d(v_i, v_j)$ from Equation 4-2 above.
Listing 4-8: Method to Calculate Velocity Differences

The implementation of the two terms of Equation 4-3 which together represent the positional differences of the joints is shown below.

Listing 4-9: Calculating the First term of the Position Differences Function

As can be seen from code Listing 4-9, the first term of the positional differences function is only considered if the frames that are being compared are from the same motion sequence. In other words, if the frames are from different motion files, the first term of Equation 4-3
is ignored. The main reason for this change to the original implementation is due to the fact that most motion files from different sources are disparate in terms of the location of the skeleton in 3D space throughout the extent of the motion sequence. Therefore, if the first term was always included irrespective of which motion files were being compared, very few matches would be found and so there would be a limit to the number of new motions that could be generated. However, although the translation data of the root joints of the two frames is ignored at the matching stage, it still needs to be factored back in when the actual motion is created, so that the translation data produces a smooth looking motion transition. Listing 4-10 below illustrates how the second term of Equation 4-3 is implemented in code. Once again, the quaternion operations discussed earlier are extremely useful.

```c++
//Calculate secondTerm
Channel::Names fromnames = from->getNames();
//Channel * fromchan;
//Channel * tochan;
for (Channel::Names::iterator k = fromnames.begin (); k != fromnames.end (); k++)
{
    Channel * fromchan = from->getChannelEntry (1, *k);
    Channel * tochan = to->getChannelEntry (1, *k);

    //std::cout << "Name = " << *k <<"\n";
    //std::cout << "Weight = " << findJointWeight("X") <<"\n"

    Quaternion S = tochan->rot;
    Quaternion R = (S.Inverse ()) * fromchan->rot;
    Vector JointDiff = Log(R);
    // std::cout << "JointDiff = " << JointDiff <<" 1 = " << 1 << " j = " << J <<"\n"
    SecondTerm = SecondTerm + (findJointWeight("X") * pow((JointDiff.length()),2)));
}
```

**Listing 4-10:** Calculating the Second term of the Position Differences Function

### 4.5.3 Fixed and Relative Coordinate Systems

According to Lee et al (2002), position and orientation of the root segment of the skeleton can be represented in a fixed (world) coordinate system, or as a relative translation and rotation with respect to the previous frame of the motion. They go on to state that with a fixed coordinate system, transitions will only occur between motion sequences that are located nearby in three-dimensional space, while the relative coordinate system allows transitions to similar motion recorded anywhere in the capture region. As already mentioned, when comparing frames from different motion files, the translational position
of the skeleton was ignored and thus the relative coordinate system was used. Then, when generating the actual motion, a relative translation value is calculated by comparing the changes in the root position between the two motions and this is used as the new translation value for the root joint.

### 4.5.4 Constructing and Pruning the Motion Database

The above sections describe how the Markov process equations were implemented for this project and what changes were made to them. This section relates to the data structures that were used to store the results of applying the Markov process. First of all, we identify the main steps in the motion synthesis process. (Figure 4-3 below)

![Figure 4-3: Motion Synthesis Process](image)

At this stage, it is important to emphasize the distinction between the motion archive and the motion database. I have used the term **motion archive** to describe the collection of raw data that has not yet been processed using the Markov process. On the other hand, the term **motion database** has been used to describe the summarized results of applying the Markov process to a subset of the motion archive.
The resulting matrix of probabilities is extensive and requires a large amount of storage space. Since most of the frame to frame permutations result in low probabilities, Lee et al (2002) suggest pruning the database to reduce the storage space required and improve the quality of the resulting motion by avoiding too frequent transitions. Pruning is done based on a number of criteria (already discussed in Section 2.1.2). Only two methods have been implemented to prune the database at present. The first is to ignore all transitions with probabilities below a user-specified threshold. This threshold is set by the user just before the preprocessed motion database is created. The other method is to ignore all transitions that occur between frames in the same motion file where the “from” frame is greater than the “to” frame. Not only does this reduce the number of possible transitions, but also it ensures that the dynamics of the motion before and after the transition remain valid. For example, this will eliminate the possibility of a skeleton walking forward in one frame and then walking backward in the next.

However, instead of creating a matrix of all possible transition probabilities and then pruning it at a later stage, a vector list is used to keep track of all transitions that pass the pruning criteria mentioned above. The structure of this vector list is shown below:

```c
typedef struct framesprob {
    int from, to;
    double prob;

    friend bool operator<(const struct framesprob &a, const struct framesprob &b) {
        return a.prob < b.prob;
    }
} framesprob;

typedef std::vector<framesprob> FramePairs;
```

**Listing 4-11:** Vector List representing Results of the Markov Process

As can be seen from the code above, each valid transition is stored in terms of the frames involved as well as the probability associated with the transition. Also, since the motion database can be constructed from several different motion files, the “from” and “to” frame numbers represent the corresponding, accumulated frame numbers. Therefore, to be able to extract the motion file that a particular frame number is part of, it is necessary to make use of a second vector list. Its structure is shown below:
Listing 4-12: Vector List of Motion Files and Frame Limits in the current Motion Database

The above list stores the lower limit and upper limit of frame number as well as the associated motion file. Joint weightings and the threshold value specified by the user before the Markov process was run are also stored in this vector list.

By making use of these two vector lists, the user has the option of creating a number of different motion databases and then saving them to file so that the frame matching process does not have to be repeated each time the program is restarted. Each database consists of a list of the motion files and the results of applying the Markov process to these motion files. In other words, the contents of the above vector lists are written to a file which forms the current motion database.

This representation makes it possible to add new motion files to the motion database at any time by simply updating the vector lists for the current motion database. When the new motion is generated, it is these vector lists that are used to identify all the possible transitions.

4.5.5 Motion Control

At present, there are three main methods that are available to the user in order to control the synthesis of new motion:

- The first option allows the user to specify which motion file from the current motion database will form the base motion for the synthesis process. Initially, the best possible transitions are chosen when the base motion is changed. What this essentially does is
to display the original motion of the base motion file. All other motion synthesis operations are then performed relative to the base motion file. For example, if the motion database has two motions, say a basketball dribble motion and a basketball dunk motion, it makes more sense to create a new motion that starts with part of the dribble motion and then transitions to the dunk motion. In this case, the dribble motion is set as the base motion for the synthesis process.

- With the second option, new motion is generated by randomly selecting a possible transition from those options stored in the vector lists representing the current motion database. In this way, the new motion sequence begins with frame 0 from the base motion and then consists of a randomly selected combination of motion segments from the underlying raw motion data.

- The last option allows the user to select which transition will be taken next. The user specifies the threshold value and then as the motion is displayed, the best possible transition options are displayed on screen so that the user can choose one. This option represents the implementation of the choice interaction technique described earlier in the thesis.

### 4.5.6 Blending Transitions

In order to improve the quality of the synthesized motion sequences, a method of blending transitions has been implemented. Joint angle interpolation is used to generate a user specified number of extra frames that help smooth the transition that occurs from one motion to another. The results of this blending technique are discussed in the next Chapter.

### 4.5.7 Removing Redundant Frames

Schodl et al (2000) point out that certain transitions might lead to a portion of motion that has no exits (such as the end of one motion file). This implies that several frames towards the end of the newly created motion will be identical and thus no movement will result. Although Lee et al (2002) suggest an approach to avoid such dead ends, it is fairly costly in terms of processing time, and so a simpler solution has been adopted. No attempt is made
Chapter 4: Implementation

Interactive Synthesis of Avatar Motion from Preprocessed Motion Data

to avoid dead ends, but when they do occur, a count is kept of the number of repeating frames. The user is then given the option to crop the synthesized motion sequence by eliminating the redundant frames at the end of the sequence.

4.6 Exporting to Blender

As discussed in Section 3.1, one of the main requirements of the system is that the user should be able to export the newly created motion to Blender so that additional modelling can be done using the Blender interface. Therefore, a python script that reads in a BVH file, sets up a skeleton structure and generates its associated motion curves needs to be developed. Due to time constraints and a lack of knowledge about the Python scripting language, it was necessary to look for an existing python script on the Internet. Fortunately a script developed by McKay (2001) was discovered. The script parses a BVH file, extracts bone and motion data and then generates the skeleton hierarchy and motion curves. Unfortunately, the script only seems to work with Blender version 2.23 because of syntax changes that were made to the python scripting interface used by more recent versions of Blender. Other limitations of the script are:

- The parser ignores the end effector information stored by the joints that are named using “End Site”.
- In Blender, a skeleton structure is built up from a series of IKA (Inverse Kinematic Animation) chains. It is up to the user to set up these chains so that the resulting hierarchy is the same as the one in the file being imported into Blender.

The lack of end effector information is not a major problem due to the fact that there is no motion data associated with the joints that are labelled “End Site”. This information merely defines the length of the bones at the end of each branch in the skeleton hierarchy. The second limitation implies that the hierarchy of the skeleton structure built up in Blender, including identical joint names, must match that of the exported motion file skeleton structure.

The snippet of code shown next, illustrates how Blender is called from within the motion synthesis tool. First, the current skeleton and associated motion is written to a file called
temp.bvh in the current directory and then Blender is run using the default .blend file supplied by McKay (2001). This file has been modified slightly in order to automatically start the python script by providing it with the location of the temp.bvh file.

```cpp
if (sv->m->exportBVH)
{
    writeBVH ("temp", skel, newmov, sv->z->frameslider->maxValue|+1);
    std::system("blender -v \"default.blend\"");
    sv->m->exportBVH = false;
}
```

**Listing 4-13**: System call to Open Blender

The results obtained from using this export script are discussed in more detail in the next chapter.

### 4.7 Summary

This chapter has dealt with the main implementation issues involved with developing the system that was designed in Chapter 3. A combination of OpenGL, Qt windowing toolkit, C++ and the Blender Python Scripting Language was used to implement the system on the Linux RedHat 8.0 operating system platform. The implementation of each component of the system was then discussed with added emphasis on the motion synthesis component.
5 Experiments and Results

Having described the design and implementation of the proposed system in the previous two chapters, all that is left to do is discuss the results that were achieved. The following chapter is divided into several sections; each section describing the various achievements associated with each project objective identified in Chapter 1. An evaluation of the motion synthesis approach taken is then given and a number of graphs are shown to support some of the conclusions made.

5.1 Motion File Archive

The first project objective involved creating an archive of motion capture data in a common format. At present, I have built up an online archive of raw motion data consisting of approximately 300 BVH files in 11 different categories ranging from dancing to death. The quantity and range of BVH files collected thus far proves that the BVH file format is in fact very popular and given more time an even larger motion archive could have been compiled. A shell script has been written that automatically generates the HTML code to update this archive. The skeleton structures defined by the various BVH files vary considerably from one source to another which supports the need for the motion file converter.

5.2 Motion File Converter

To ensure that all the BVH files can be compared using the Markov process, a BVH converter has been implemented to standardize the motion files to use a maximal skeleton structure. As discussed in the previous chapter, two methods have been developed to implement the conversion process. A comparison of these two conversion techniques is given next, as well as an outline of the problems encountered with using a combination of quaternions and Euler Angles.
5.2.1 A Comparison of the Two Conversion Techniques Used

When the user chooses to convert a BVH file to one that uses the maximal skeleton hierarchy, they are presented with the option shown by the screen shot below:

![Conversion Technique Dialog](image)

**Figure 5-1: Conversion Technique Dialog**

The first option relates to the first method discussed in Section 4.3.3, which describes the process of modifying the joint angles of the original BVH file to match the initial skeleton pose of the maximal skeleton hierarchical structure. Since it is the actual orientation values that are modified, it is necessary to make use of quaternion arithmetic. This implies that the Euler Angles from the original file must be converted to quaternions, modified and then converted back to Euler Angles, before the new BVH file is written to file. Unfortunately the final conversion back to Euler Angles introduces a few errors which will be explained in more detail in Section 5.2.2. It was due to these errors that a new conversion technique, that does not need to make use of quaternions, was developed.

Before discussing this alternative method, a screen shot illustrating the results of using the first method is shown over the page. The BVH Converter Tool was used to convert a BVH file called walk1.bvh to a BVH file with a maximal skeleton hierarchy. As seen in Figure 5-2, the tool consists of two main widgets. The top widget is where the results of the conversion process are displayed. In this case, the lighter looking skeleton (green skeleton) represents the skeleton hierarchy described by the original BVH file while the darker skeleton (red skeleton) represents the maximal skeleton hierarchy. The bottom widget contains all the controls options available to the user. Apart from the standard playback and camera controls, the user is also able to save the converted BVH file.
The second option relates to the second method discussed in Section 4.3.3, which describes the process of modifying the bone lengths of the original skeleton structure to represent the maximal skeleton structure. This is done by modifying the offset values associated with each joint in the skeleton tree. The joint angles representing the orientation of each bone are not modified at all, which means that there is no need to make use of quaternions. Although there are no errors introduced due to quaternion to Euler conversions, the implementation is still not quite right. If a wire-frame representation of the converted skeleton is used, then the converted hierarchical structure is perfect (similar to the one shown in Figure 5-2), but with some skeleton hierarchies, the solid rendering option produces inaccurate results (shown in Figure 5-3). This implies that although the conversion process is correct, the method used to render the bones is not perfect.
Unfortunately, there was not enough time to investigate this implementation error any further, but can be added to the list of possible extensions.

5.2.2 Converting Quaternions to Euler Angles and Vice Versa

As already mentioned, there are still some serious errors in the formulae used for converting the modified quaternions back to Euler Angles, so that the new motion data can be written to file. Several slightly different formulae have been used in an attempt to get round this issue. Unfortunately, no matter what formula was used, the results still included small errors. There are a number of possible causes of these errors.

Firstly, the problem may occur due to the fact that a Euler rotation has many representations. For example the combination of Euler Angles (0,0,0) is identical to (180,180,180). Strout (2003) confirms that this is a problem by stating that the conversion process from quaternion to Euler Angles is not unique and that this lack of uniqueness can cause big problems both computationally and conceptually.

Another possible cause of the errors could be due to the fact that the conversion process is sensitive to round-off errors. Some of the intermediate results in the conversion process may loose precision at some stage or another which leads to the inaccurate Euler Angles at the end of the conversion.
Once again, there was not enough time to fully investigate the source of these errors and so although a number of different conversion formula were tested, the problem still persists and thus can also be added to the list of possible enhancements to the tool.

5.3 Motion Display and Synthesis Tool

The results and achievements associated with Objectives 2 and 3 from the first chapter have been grouped together under this section. Objective 2 was to develop a tool to animate the motion data extracted from the BVH files using a graphics library such as OpenGL. Objective 3 was to investigate and implement an approach to synthesize new motion sequences from existing motion data given some simple user input.

A simple motion display tool has been developed. This will help the animator control the motion synthesis process and view the results of combining different motion files. Additional functionality such as zooming, rotating and translating the 3D view using the mouse has been added to the interface to help the animator view the motion from the best angle and position. The screen shot over the page, shows the main interface components of the motion display and synthesis tool. As seen in Figure 5-4, a playback control panel has also been created which allows the user to playback, loop and stop the current motion. Three camera control buttons have been added; the first resets the camera to point at the origin (0,0,0), the second camera control called “Glue” mounts the camera in its original position, but rotates it so that it is always looking at the root joint of the skeleton and lastly, the third option “Track” translates the camera so that the root joint of the skeleton is always in the centre of the screen.

To help the user conceptualize the skeleton structure of the current base motion file, a tree view of the skeleton’s hierarchy is built up and can be seen on the right of the display widget. Clicking on the different bone names in the tree view highlights the selected bones in the display area.
Based on the Markov process explained by Lee et al (2002), the motion display tool has been extended to allow the user to create new avatar motion. The administration features provided by this enhancement include:

- Ability to create a new Motion Database File. When a new motion database is created, the user selects an initial base motion and specifies the threshold value to be used during the Markov process. The user is also able to specify the weights of different joints for the Markov process. The initial base motion essentially sets up the skeleton structure that all additional motion files added to this database must define. Figure 5-5 illustrates the dialog used to capture the Markov Process settings.
Chapter 5: Experiments and Results

Interactive Synthesis of Avatar Motion from Preprocessed Motion Data

Figure 5-5: Dialog Used to Request Markov Process Settings.

- Ability to load an existing Motion Database that has already been processed.
- Allows the user to save a newly generated Motion Database. This means that the Markov process does not have to be repeated the next time the database is needed.
- Allows the user to save the current BVH motion file.
- The user can add new motion files to the current motion database and thus update the associated list of possible transitions. Before the new motion file is processed, its skeleton hierarchical structure is compared to that of the current base motion. If they are different, the process is aborted and the following error message is displayed:

Figure 5-6: Incompatible Skeleton Hierarchy Error Message.

- Allows the user to change the base motion thereby generating a new starting frame for the current motion.

The other main features of the motion synthesis tool are described in the following sections:

71
5.3.1 Motion Control

The three main options that have been provided to control the motion synthesis process have already been discussed in Section 4.5.5. The first option involves selecting a base motion file from the current motion database. This essentially sets up the original motion sequence associated with that motion file. This option is used as a starting point for the rest of the motion synthesis process. The second option allows the tool to randomly select which transitions to use to build up a new motion sequence. The screenshots below illustrate a transition point discovered using the second option, by applying the Markov process to a motion database consisting of a drunk walk motion and a cool walk motion using a threshold value of 0.9. The screenshot on the left shows the frame from the cool walk motion and the screenshot on the right shows the frame from the drunk walk motion.

![Figure 5-7: Two Consecutive Frames Illustrating a Transition Point.](image)

The third option allows the user to select which transitions will form part of the newly generated motion. The “Choose Transitions” option is selected and the base motion is played back. Whenever there is more than one possible transition from the current frame, a list is displayed showing all the options. When the user highlights a particular transition, a preview of the transition is shown in the display widget below the list. In this way, the user can select exactly which transitions will make up the new motion. The screen shot over the page shows an example of the possible transition list and the preview widget that is displayed when the choose transition option is selected using a motion database consisting of basketball dribble and dunk motions.
The user can also set the current threshold value at any time during the motion synthesis process, without having to process the motion database again, as long as the value is greater than the original threshold value used when the motion file was added to the database.

### 5.3.2 Blending Transitions

As already explained in the previous chapter, a form of joint angle interpolation was used to create in-between frames at transition points in order to make the new motion smoother. When the blending option is chosen, the user specifies how many in-between frames they would like the tool to insert. The blending option has been implemented so that interpolation is only done for transitions that occur between two different motion files, not within the same motion file. The two screen shots over the page illustrate two consecutive frames in a newly generated motion. The one on the left is from the basketball dribble...
motion and the one on the right is from the basketball dunk motion. As can be seen from the bone positions in each frame, there is quite a big difference between the two. This difference creates a jerky resulting motion.

![Figure 5-9: Two Consecutive Frames without Interpolation.](image)

The following sequence of screen shots represents six consecutive frames that have been generated using the same motion files but with blending enabled. The first and the last frames are identical to the ones shown above. The tool has automatically inserted four additional frames that help to smooth the transition that occurs between the first and the last frames.
5.4 Exporting to Blender

The use of the python script developed by McKay (2001) has already been established in the previous chapter. All that is left to do is evaluate the export function it provides and display some of the results obtained. At present, although the default skeleton structure generated by the script does not represent the maximal skeleton structure exactly, this can be altered by adding extra IKA chains to the default file. The script seems to perform very well as illustrated by the screen shot below. Thus, the export function will be very useful in that new motion can be generated using the motion synthesis tool and then exported to Blender so that additional modelling and skinning operations can be done to improve the quality of the animation and model.

Figure 5-11: Exported Motion in Blender.
5.5 Evaluation of the Motion Synthesis Process

In order to help analyse some of the statistics that can be obtained from applying the Markov Process to different motion databases, a graphing wizard tool has been created. Essentially the wizard is capable of producing four main graph types as shown by the screen shot below.

![Figure 5-12: Screen shot illustrating possible Graph Types](image)

The first two options allow the user to generate graphs representing the number of possible transitions at each frame, either for a single motion sequence or for two motion sequences, so that a comparison may be done. Additional options can be set relating to the motion file to use, the threshold, the plotting colour, the graph title and the curve style (shown in Figure 5-13). Figure 5-14 shows a sample graph generated by using the second option of the graphing wizard with curve style unique.

![Figure 5-13: Graph Options](image)
Chapter 5: Experiments and Results

Interactive Synthesis of Avatar Motion from Preprocessed Motion Data

Figure 5-14: Graph comparing two motion files based on their Possible Transitions.

The previous graph illustrates the main segments of the two motion files where the most transitions occur. The transitions considered here can be to other motion files or within the same motion file. Being able to generate these statistics will help the animator decide whether the current threshold value is too high or too low for the required accuracy of the motion synthesis process. The next graph shows a comparison of the same two files, but instead of using the unique curve style, a Bezier curve style is used instead:
The third graph option in the graphing wizard draws similar graphs to those shown above, however, only the transitions that occur between different motion files are plotted. This is extremely useful for the animator as it allows them to check in advance exactly where in the motion sequence the main transitions to other files occur and how many transition jumps there are. For example, if the animator wishes to create a motion file that consists of part of the basketball dribble motion and part of the basketball dunk motion, by using the choice interaction technique, they may wish to know in advance where the main transition jumps between the two files occur. The graph over the page illustrates the statistics required:
Figure 5-16: Main Transition Jumps between dribble and dunk Motions.

The final graphing option allows the user to plot the orientation values relating to specific joints of the current skeleton hierarchical structure. Graphs of this type will help the user identify which joints may be causing artefacts in the newly generated motion. They may also be useful in identifying how many in-between frames should be added during the interpolation process in order to produce a smoother looking motion. A sample graph, illustrating the orientation values for the hips, is shown over the page in Figure 5-17. An alternative to using the graphing wizard to generate similar graphs, the motion curves automatically generated by the python script when a BVH file is exported to Blender can also be used to analyse individual joint rotation values.
Chapter 5: Experiments and Results

Interactive Synthesis of Avatar Motion from Preprocessed Motion Data

Figure 5-17: Sample Orientation values for the Root Joint.

A number of tests have been carried out to establish whether the implemented approach to the motion synthesis process is capable of producing realistic looking motion. Several motion databases have been created and a number of graphs similar to the ones illustrated above have been produced to help evaluate the approach. It was found that the Markov Process that was implemented is in fact extremely effective in identifying good transitions between different motion sequences. However, as mentioned earlier, the size and quality of the underlying motion archive is key to the success of the implementation of Lee et al’s work. Thus, although reasonable transitions were found between different motion files, it was necessary in most cases to use a low threshold value of about 0.6 for the process to find them. Therefore, in order to provide the possibility of better transitions between motion sequences, the size of the underlying motion archive should be increased considerably. Luckily, since there are a number of different sources of free BVH files online, compiling a larger motion archive would not be too difficult, given enough time.
5.6 Summary

This chapter has provided a number of screen shots illustrating some of the main features and aspects of the tools developed during the implementation of this project. The main deliverables are linked back to the project objectives that were identified in Chapter 1 and finally, an evaluation of the implementation of the motion synthesis process used in the project was given. The final chapter concludes the project work and identifies areas for future work.
6 Conclusion

The following chapter provides a brief discussion and summary of the project on a whole as well as a description of the projects main accomplishments and pitfalls. Finally, some possible areas for future work or enhancements to the developed tools are suggested.

6.1 Discussion and Summary

In this thesis, the main design and implementation issues involved with developing a motion synthesis tool, similar to the one created by Lee et al (2002), are discussed. To gain a better understanding of the motivation behind the project, the thesis begins by establishing the goals of the project as well as identifying some of the related research being done on avatar animation and control in general. Emphasis is placed on the techniques used to edit already captured motion data in order to make it reusable. The motion synthesis tool developed here uses a first order Markov process to establish possible transitions between frames in different motion files from the underlying motion archive. In this way, new motion can be synthesized by blending different segments from various motion files. For these motion files to be compared, they need to describe motion sequences for a common hierarchical character structure. Unfortunately, due to the diversity of motion files collected from different sources on the Internet, there was no match in the hierarchical structures. This meant that it was necessary to create a motion file converter to ensure all the motion files used in the Markov Process described motion for a common skeleton structure. In order to test the effectiveness of the developed motion synthesis tool, a number of new motion sequences were developed from the underlying motion archive and evaluated in terms of how realistic and natural the movements were. It was established that although the approach used is relatively accurate in identifying good transition points between motion files, the success of the approach mainly depends on the size and quality of the underlying motion archive. In this case, if the motion archive were to be extended, the results obtained would have been even more impressive.
6.2 Accomplishments and Limitations

The main accomplishments of this project include the following:

- A fairly large motion archive consisting of a number of motion files collected from several different sources.
- A motion file converter that ensures that any two motion files describing motion for different characters can be compared using the Markov Process.
- A motion display and synthesis tool, capable of animating an avatar and providing the user with an intuitive interface for generating new motion sequences by blending motion segments from motion files found in the motion archive.
- An option within the motion synthesis tool that enables the newly created motion sequences to be exported to Blender where additional modelling and animation techniques can be applied to the avatar to improve the quality of the resulting motion sequence.

Although the project was a great success overall, there are still some limitations that need to be mentioned:

- The process used to convert Quaternions to Euler Angles introduces a number of precision errors which implies that when the new motion sequences are stored to file, the movement of the avatar may not be accurate.
- The techniques used to render the solid representation of the avatar are not accurate after the motion file has been converted to represent motion for the maximal skeleton hierarchy.
- Lastly, no actual frames are created except for the in-between frames generated during the interpolation process. The only way new motion is synthesized is by cutting and pasting different parts of motion files together.

6.3 Future Work

As mentioned previously, there is a lot of research that is being done within the area of motion editing techniques. Thus there are a number of extensions that could be made to
the tools that have been developed here. Some of the possible extensions include the following:

- Work could be done on improving the Quaternion to Euler conversion process in order to eliminate the precision errors that occur at present.
- Similarly, improvements could be made to the rendering techniques currently implemented.
- Experiments could be done to test to see if motion for avatars could be applied to animal hierarchical structures. This would probably involve developing a tool that allows the user to easily change the maximal skeleton hierarchy used in the motion display and synthesis process.
- Additional motion editing techniques could be implemented in order to support and enhance those already provided. For example, one option is to plot the path of the avatar over the entire motion using splines, based on the translation and rotation data associated with the root node. The user would then be able to manipulate the path taken by the avatar by moving some of the control points of the splines.
- The other interaction techniques described by Lee et al (2002) could be investigated and implemented if found to be viable interfacing options.

Thus, although the implementation of the approach described by Lee et al (2002) (for the interactive synthesis of avatar motion based on some simple user input) has been successful, there are still a number of areas in which the developed tools can be enhanced in order to improve the overall results of editing the motion capture data and investigating these areas could be a worthwhile endeavour.
Index

ASF/AMC, 20
avatar, 2
base motion, 61
bone, 34
BVH, 20, 24
Channel, 34
cluster analysis, 14
cluster forest, 14
cluster tree, 14
clustering. See cluster analysis
clusters, 14
controller, 16
degree of freedom (DOF), 34
Euler Angles, 46
gimbal lock, 46
graph walk, 17
HTR, 21
IKA, 63
key framing, 3
maximal skeleton hierarchy, 26
motion archive, 29, 59
motion capture, 3
motion database, 59
motion graph, 10, 17
OpenGL, 41
physically based modelling, 3
quaternion, 46
skeleton, 34
spacetime constraints, 17
<table>
<thead>
<tr>
<th>Reference</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference</td>
<td>Title</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Reference</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Reference</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
Appendix A – Shell Script to Generate HTML

#!/bin/bash

# Create HTML for Resources Page
readestfile="/mnt/windows/WebStuff/ResearchProject/resources.html"
tgsrc="/mnt/windows/WebStuff/ResearchProject/assets/Resources/HttpServer/AllFiles/" tgsybsrc="/mnt/windows/WebStuff/ResearchProject/assets/Resources/HttpServer/AllFiles/"
count=0

rm $readestfile

echo "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">" >>$readestfile
echo "<html>" >>$readestfile
echo "<head>" >>$readestfile
echo "<title>Resources</title>" >>$readestfile
echo "<meta http-equiv="Content-Type" content="text/html; charset=utf-8">" >>$readestfile
echo "</head>" >>$readestfile

echo "<body bgcolor="#000000" text="#9999FF" link="#00FFFF" vlink="#009999"">" >>$readestfile

echo "<div id="navstrip">" >>$readestfile
echo "<div id="navstrip">" >>$readestfile

echo "The following page provides a collection of motion capture data that I will be using for testing the system developed during my Honors Project research. " >>$readestfile

echo "Motion Capture" "(font color="#00FFFF"" >>$readestfile

echo "files are grouped into various motion categories and can be downloaded by following the links below:" >>$readestfile

echo "The files provided are all in the Vicon's Hierarchical Data Format (" >>$readestfile

echo "files can be found at http://www.cs.indiana.edu/graphics/Courses/cs-838-1999/EVH.html" >>$readestfile

echo "<p>" >>$readestfile

echo "<div align="center">Motion Capture Database</div>" >>$readestfile

echo "<p align="center">Click on one of the motion categories below to view a list of available motions:</p>" >>$readestfile

cd ResearchProject/Assets/Resources/EVHSampleFiles

for dir in $files

do
echo "<a href="/dir.html">" >>$readestfile

echo "&lt;font face="Comic Sans MS">&lt;font" >>$readestfile

echo "<a href="/dir.html">" >>$readestfile

done

echo "</body>" >>$readestfile

echo "</html>" >>$readestfile

# Create each Motion Category Page

for dir in $files

do
count=0

echo "<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">" >>$destfile

echo "<html>" >>$destfile

echo "<head>" >>$destfile

echo "<title>$dir Motion Capture</title>" >>$destfile

echo "<meta http-equiv="Content-Type" content="text/html; charset=utf-8">" >>$destfile

echo "</head>" >>$destfile

echo "<body bgcolor="#000000" text="#9999FF" link="#00FFFF" vlink="#009999"">" >>$destfile

echo "<div align="center">" >>$destfile

echo "<div align="center">" >>$destfile

90
Appendix

Interactive Synthesis of Avatar Motion from Preprocessed Motion Data

Loop through each source in the motion directory and create separate tables.

```bash
cd $dir
for site in `ls`;
do
cd site
  count=[$count]
  echo going into dir: $site
  /SourceInfo.txt $destfile
  echo "<table width="99%" border="1">" >>$destfile
  echo "<tr bgcolor="#0000FF">" >>$destfile
  echo "<td width="16%" height="25" > File Name </td>" >>$destfile
  echo "<td width="16%" height="25" > File Description </td>" >>$destfile
  echo "<td width="16%" height="25" > File Type </td>" >>$destfile
  echo "</tr>" >>$destfile
  # Loop for each BVR file in source directory for file list (ls *.a3d)
do
    echo "<tr>" >>$destfile
    echo "<td><a href="/Assets/Resources/BVRsampleFiles/dir/$file/file.html">$file</a></td>" >>$destfile
    echo "</td><a href="/Assets/Resources/BVRsampleFiles/dir/$file/file.html">$file</a></td>" >>$destfile
    echo "<td align="center">BVR</td>" >>$destfile
  done
  echo "</table>" >>$destfile
  echo "<br>" >>$destfile
  echo "<hr>" >>$destfile
  rm $getBySource/$dir-$site.tar.gz
  cd ..
  cd $dir-$site.tar.gz
tar cf $getBySource/$site.tar.gz
  # Download all $dir motion files for source count "<a href="/Assets/Resources/zipperedMotionBySource/$dir-$site.tar.gz" target="_blank">here</a>" >>$destfile
  echo "<br>" >>$destfile
  echo "<br>" >>$destfile
done
echo leaving $site
rm $getBySource/$dir-$site.tar.gz
cd ..
```

# echo "&nbsp;" >>$destfile
# echo "&nbsp;" >>$destfile
# echo "&nbsp;" >>$destfile
# echo "&nbsp;" >>$destfile
# echo "&nbsp;" >>$destfile
# echo "&nbsp;" >>$destfile
# echo "&nbsp;" >>$destfile
# echo "&nbsp;" >>$destfile
# echo "&nbsp;" >>$destfile
# echo "&nbsp;" >>$destfile
```
Appendix B – Project Source Code

The project source code at present is several thousand lines long and thus has not been included in hard copy format. However, the code can be found in full on the supplied project CD.

Appendix C – Source List

<table>
<thead>
<tr>
<th>Source</th>
<th>Site</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Michael D’Andrea</td>
<td><a href="http://centralsource.com/blender/bvh/files.htm">http://centralsource.com/blender/bvh/files.htm</a></td>
<td></td>
</tr>
<tr>
<td>Dr. Ron Metoyer</td>
<td><a href="http://web">http://web</a>. engr. oregonstate. edu/~zier/Assignment4BVH/</td>
<td>EECS Department, Oregon State University</td>
</tr>
<tr>
<td>Credo Interactive</td>
<td><a href="http://www.charactermotion.com/trial/freemocapform.htm">http://www.charactermotion.com/trial/freemocapform.htm</a></td>
<td>New motion capture files are available every month – must supply email address.</td>
</tr>
<tr>
<td>Help 3D</td>
<td><a href="http://www">http://www</a>. Help3d.com</td>
<td>Login as guest to gain access to free MoCap.</td>
</tr>
<tr>
<td>Motek Entertainment</td>
<td><a href="http://www.e-motek.com/members/stockmoves">http://www.e-motek.com/members/stockmoves</a></td>
<td>The umf converter introduces a bracket error in the BVH file, but this can be easily fixed after the conversion.</td>
</tr>
</tbody>
</table>

Files are downloaded in .umf format. Use the provided umf converter to convert them to .BVH format.