A Motion Graph Approach for Interactive 3D Animation using Low-cost Sensors

Mithilesh Kumar

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Masters of Science
in
Computer Science and Applications

Yong Cao, Chair
Doug Bowman
Dane Webster

July 14, 2008
Blacksburg, Virginia

Keywords: Performance animation, accelerometer, motion graph, interpolation, motion synthesis
Copyright 2008, Mithilesh Kumar
Interactive 3D animation of human figures is very common in video games, animation studios and virtual environments. However, it is difficult to produce full body animation that looks realistic enough to be comparable to studio quality human motion data. The commercial motion capture systems are expensive and not suitable for capture in everyday environments. Real-time requirements tend to reduce quality of animation. We present a motion graph based framework to produce high quality motion sequences in real-time using a set of inertial sensor based controllers. The user’s action generates signals from the controllers that provide constraints to select appropriate sequence of motions from a structured database of human motions, namely motion graph. Our local search algorithm utilizes noise prone and rapidly varying input sensor signals for querying a large database in real-time. The ability to waive the controllers for producing high quality animation provides a simple 3D user interface that is intuitive to use. The proposed framework is low cost and easy to setup.
Dedication

To my Mom and Dad for the wonderful education they have given me
To my family and friends, who have supported me always
To the thirty two Hokies who lost their lives on
16th April, 2007
Acknowledgments

It has been an amazing learning experience for me at Virginia Tech, and working on this thesis was no different. I have had the opportunity to work with the best minds, collaborate with several teams and get help and insightful suggestions. This thesis would have been incomplete without the help of many people to whom I would like to express my deep and sincere gratitude.

I would like to thank my advisor Dr. Yong Cao, for his guidance, inspiration, support, and his friendship. It has been an amazing learning experience working along with him. He has always motivated me to think deep and to question every idea. He has provided me with all the resources required for this work - in the form of ideas, equipments and graduate assistantship.

I would like to express my sincere thanks to my committee members Dr. Doug Bowman and Dr. Dane Webster for their regular guidance and diverse thoughts. Any conversation with them has inspired me to think differently. Their views have had a remarkable influence on my research and their comments helped me shape this thesis.

Dr. Eli Tilevich has helped me understand the software engineering principles that made the development process of my thesis much simpler and efficient. Without his help, it would have been difficult to achieve my objectives. My warm and sincere thanks to him for all the help and support that he has extended to me.

I would like to thank the Computer Science Department that has supported me with graduate teaching assistantships and support with various resources. I will never forget the pleasure of meeting the administrative staff who have helped me with a smile on their face and eagerness in their heart.

Thank you Chreston Miller, Sean Ponce and Ashley Robinson with your beautiful voice and tremendous performances. Without your help, I would have not been able to make the wonderful videos that have helped me reach out to a wider audience.

Most importantly, I would like to thank my family and friends for their love and for being supportive throughout. They have always been by my side and have made me the person that I am proud to be today.
1 Introduction
   1.1 Motivation .......................................................... 1
   1.2 Contributions ...................................................... 3
   1.3 Outline .............................................................. 5

2 Related Work .......................................................... 6
   2.1 Interactive Animation Control .................................... 6
   2.2 Data-Driven Animation using Motion Capture ...................... 9
      2.2.1 Synthesis by concatenation ................................... 9
      2.2.2 Statistical modeling .......................................... 11
      2.2.3 Motion interpolation ......................................... 12
   2.3 Software architecture for 3D animation pipeline ................. 13
      2.3.1 Pipes and filters ............................................. 13
      2.3.2 Layered design pattern ..................................... 14
      2.3.3 Mixin layers .................................................. 15

3 System Architecture and Implementation ................................ 16
   3.1 Data Collection and Representation ................................ 16
   3.2 Motion Generation .................................................. 18
   3.3 Application software design and implementation .................. 18

4 Data Collection and Representation .................................... 22
List of Figures

3.1 Data collection using Vicon motion capture system and Nintendo Wii Controllers. Ultimately, all collected data is represented as a Motion Graph. (Image by author) .............................................................. 17
3.2 System architecture for real-time motion generation. (Image by author) .... 19
3.3 The pipes and filters architecture ............................................................. 20
4.1 Data collection: an optical motion capture system and a 3D acceleration sensor based data acquisition system are used in parallel. There are 45 retro-reflective markers and eight sensors (four WiiTMNintendo controllers) attached to the performer. (Image by author) .............................................. 24
4.2 An example motion graph generated from two sets of motion capture data . 28
4.3 The distance matrix for two sample clips - the red spots have been identified as transition points ................................................................. 30
4.4 The simplified motion graph structure shows (a) before merging source nodes and (b) after merging source nodes ........................................... 31
5.1 Default pose - an arm lifting action is more likely to transition to (a), golf to pose shown in (b) and tennis, basketball to (c) .............................. 36
5.2 The default motion sub-graph ................................................................. 37
5.3 Creating transition from node $i$ to node $j$ by blending frames in a window .... 39
6.1 Sensor readings from the right arm sensor for tennis forehand (left) and tennis backhand (right) ................................................................. 46
6.2 Sensor readings for (a) left arm dumbbell lift, (b) right arm dumbbell lift, (c) left arm lift, (d) right arm lift and (e) tennis forehand. The top row shows readings from left sensor while the bottom row shows readings from right sensor. 47
A.1 Software implementation consists of modified version of open-source Darwiin Remote, 2 pipelines for data collection and online search and a 3D animation renderer.

A.2 Five different actions (one in each row) generated by our system. Each frame shows on the left side the actual pose and on the right side the pose from the generate motion sequence. For the purpose of comparison, we have presented the results after removing the lag. *(Image by author)*
List of Tables

4.1 Composition of motion database. .............................................. 23
6.1 System Configuration ............................................................ 41
6.2 Accuracy of the online, local search algorithm .......................... 41
6.3 Stability test ....................................................................... 43
Chapter 1

Introduction

1.1 Motivation

Realistic computer animation plays a major role in the many popular feature films, video games and virtual environments. In the past decade, motion capture technologies are re-juvenating animation industry, where high quality motion data can be captured and later edited, transformed, interpolated and stored in large motion databases (e.g., House of moves: http://www.moves.com). Any well-known virtual movie characters have been brought to life by using such motions, for example movies like “The Lord of the Rings” and video games like “The Sims”.

However, unlike video cameras, motion capture systems are still rarely used outside movie and entertainment industry. Producing high quality 3D animation requires proper planning and right tools. Therefore, it’s very difficult for a regular user to generate virtual movie characters that can be shared with a large audience. There are two main reasons for that: the cost and the technology itself. The cost of a reasonable quality motion capture system
is beyond the reach of an ordinary user, especially when compared with the average cost of the consumer electronics equipment. Most of the motion acquisition technology in such systems suffers from the disadvantages such as long setup time, restrict and limited capture space and occlusion (for vision-based system).

The two most popular technologies to produce 3D animation are motion capture and keyframing. Motion capture is the process of recording actions of human actors and using the captured data to animate 3D characters. In keyframing, an artist prepares the beginning and end frames of an animation sequence and the frames in between are generated automatically. While motion capture is costly process, the biggest setback of these techniques are that they require skilled professionals and are not suitable for use in everyday environments.

Recent research in computer graphics and animation has resulted in several new techniques to produce high quality 3D animation. However, these existing approaches suffer from tradeoffs between, quality, responsiveness and ease of production. The video game industry utilizes a lot of manual labor to produce high quality animation with responsive characters. Automatic techniques are plagued with inaccurate control. A common problem with either techniques is a lack of a intuitive interface to control character’s movement, that has near-zero learning curve. The most important reason for this is that the human body has many degrees of freedom resulting in high dimensional movements. However input devices like mice and joystick are severely limited in number of dimensions they can handle. A better match for high dimension control is the human body itself, acting as an input device for controlling the behavior of another human avatar.

In this thesis we are dealing with two major problems. First, commercial motion capture is expensive, time-consuming and complex process process. There are low cost solutions, but they are difficult to use and require a restrictive environment. Secondly, providing interactivity along with high quality animation usually involves high manual labor. In this
work we would like to address these issues and find if there exists an intuitive user interface that can be used to control 3D animation in real-time, without compromising quality.

1.2 Contributions

In this thesis, we propose a low cost, real-time, motion estimation framework based on a small number of Nintendo© Wii™ Controllers [2] that are easy to attach to body and impose little or no restriction on a motion capture environment. The controllers provide an accelerometer sensor based intuitive user interface for generating full body 3D animation in real-time. The estimated motion sequence is high quality and very realistic because it is a concatenation of motion clips from commercial motion capture system. We aim to make motion capture as easy to generate humanoid animations based on motion capture data and make it applicable for a wide range of applications, from video game interfaces, animated chat-rooms to interactive character control in virtual environments like Second Life [39]. Our target applications require high quality estimation of the user’s actions rather than exact motion reproduction, hence we call our framework motion estimation.

Our data driven approach maintains the studio quality since it uses high quality motion capture data previously captured in studio. The motion capture data is used to build a structured database called a motion graph [35]. The studio session is required only once and the users of our framework can henceforth use this database to create many more clips. The sensor based controllers are extremely easy to use mainly because it depends on natural human movements for controlling animations in real-time. The only equipment required for motion estimation is a pair of accelerometers sensor controllers, e.g. Wii Controller, and therefore our entire setup is low cost and can be used in everyday surroundings.

Our approach to produce the full body animation consists of three phases. During the first
data collection phase, we invite a professional *performer* and capture motion data using a commercial motion capture system. Simultaneously, we also capture 3D acceleration data from sensors attached to the performer’s body. We use a novel buffer polling technique to create one-to-one mapped, time synchronized data. This data is used to create a large high quality motion capture database that would be used for motion synthesis.

In the second phase, we build a motion graph using the approach of Kovar et al. [35]. Unstructured data collected in studio is processed to find out the occurrence of similar looking poses in different parts of the database. Such poses allow switching between different parts of the motion database, to create a new motion sequence. These poses together define a *transition*, which is one of the basic unit for building a motion graph. A walk along any path in the motion graph represents the currently playing animation. After building the motion graph, we use commonly used techniques to prune [48, 35, 8] the graph to retain the smoothest transitions and to remove unwanted nodes. We then introduce a novel technique for compressing the motion graph to improve performance during the search phase.

The third and final phase is the motion graph search phase. Here we propose a local, online motion graph search algorithm for obtaining appropriate transitions corresponding to user actions. In this phase, a regular user can produce real-time animation by using only the accelerometer sensors. We use the signals from the sensors as input signals to search through the motion graph for an appropriate sequences of clips that together resemble the action being performed by the user. Our novel search algorithm performs a local search, with the sensor signals acting as the query to the database. We have extended work on previous motion graph techniques to include a special motion graph called the *default motion sub-graph*. If our search algorithm is unable to find an appropriate clips to play, we transition to this sub-graph. Our approach performs at real-time speeds and provides interactive character control.
While the concept of motion graphs is not new, we are able to provide an easy to use, intuitive interface for interactive character control. The interface does not suffer from occlusion that is characteristic of vision based system. The biggest challenge to using the sensors is that the wireless signals are noisy and dynamic and that makes it difficult to use them to query the database in real-time.

We evaluate our system by computing the accuracy of graph search algorithm when a user performs. We present data for motions of various actions like arm exercise, tennis strokes, golf swing and basketball. Our results indicate that high quality full body motion can be produced with good accuracy using a small number of low cost 3D acceleration sensors. With a sound software architecture design, we achieve a scalable performance in real-time, without compromising frames per second (fps).

1.3 Outline

The rest of the thesis is organized as follows. Chapter 2 presents the background, describes the related work in this area and shows the novelty of our approach. Chapter 3 explains the system architecture and software design that enables scalable performance. Chapter 4 and 5 provide the detailed description of our approach. Chapter 6 shows the results and demonstrate the accuracy of our approach. In this chapter, we also discuss the lessons learnt and some limitations of our system. We conclude the thesis with a discussion of the lessons learnt and limitations. Chapter 7 summarizes the paper and discusses future work.
Chapter 2

Related Work

2.1 Interactive Animation Control

In a 3D virtual environment (especially computer and video games) we have a variety of user interfaces for character motions control. The input devices include mice, keyboards, joysticks and other technologies such as vision based tracking system (Sony EyeToy® [4]) and inertial sensors (Wii™ Controller). Such user interfaces can provide immediate and direct control signals with limited number of degrees of freedom. Therefore, it is difficult to provide performance-driven control for complex human motions.

In order to provide high quality control, some systems track motion of the whole user body. Such systems, also used for virtual environment applications, are based on the technologies like optical tracking (active [5] or passive [3]), magnetic, exoskeleton-based, acoustic based and inertial based. Such systems were successfully deployed in entertainment industry and adopted by animation research community for animated films and video games. However, the major disadvantages that prevent such systems from being widely used are the time used
to suit up the users and the high system cost.

Some systems are developed to reduce the system cost and suit-up time by extracting part of full body motion using a limited number of sensors. Konami\textsuperscript{TM} introduced infrared sensor based games, Mocap Boxing and Police 911 [1]. The system can track hands motion and then control the animation of boxing gloves and guns. Sony’s EyeToy [4] is markerless vision-based system capable of recognizing simple gestures, such as hand-waving and boxing-punch. Freeman et al. [23] implemented a series of computer vision based user interfaces for computer games. However, none of these systems attempted to capture the full body movements. Instead, they focused on individual body parts motions.

Badler et al. [13] proposed a system that reconstructs full-body motions using four magnetic sensors and a real-time inverse-kinematic algorithm to control a standing character in some virtual environment. The system introduced a data-driven approach to address the kinematic redundancy problem.

Another system developed by Yin and Pai [57] synthesizes full-body motion within one second by using a foot pressure sensor. However, it can only generate a small range of behaviors and cannot produce motion for complex upper body movement.

Chai et al. [18] implemented a vision based system that only requires two inexpensive video cameras. With only six markers attached to a body, the system can synthesize a wide variety of human movement without a long suit-up time. The synthesized motions are very detailed, because a data-driven approach is used to query a high quality motion capture database. Similarly Liu et al. [40] use a reduced marker set for estimating human poses that is based on linear regression. However, these systems require a restrictive motion capture environment and suffer from occlusion problem of a vision based tracking system.

Recently, tangible user interfaces are adopted when synthesizing complex full-body human
or animal motions. Oore et al. [44] use a desktop input device with six degree-of-freedom to control the locomotive animations interactively. Their input device has embedded sensors that establish a coordinate frame inherent to the character. Dontcheva et al. [20] also use an acting based animation system with a tangible interface in their puppetry system to control the motions divided into several different layers.

To combine the benefit of different motion tracking sensors, several hybrid systems were built. Their goal is to improve the quality and performance, rather than cut the system cost and suit-up time.

Foxlin et. al [22] developed an indoor system that uses both acoustic and inertial sensors. By adopting acoustic signal that can measure the distance directly, the system can correct inertial drifting problem. Similarly, Bachmann [12] introduced an inertial-magnetic system that accounts for drifting by applying magnetic signal as reference.

Most recently, Vlasic et al. [53] combine accelerometer sensors, inertial sensors and acoustic sensors together to capture high-fidelity motions that are comparable to the motions captured from marker based vision systems. The system removes the restriction of constrained motion capture environments, which allow the user to be tracked almost “everywhere”. However, the cost of the system is still high and it is not a real-time system because of the necessary post-processing time.

In this section, we have seen two patterns. There are devices that are easy to use, but produce low quality animation. On the other hand, we have devices that producing good quality animation but are not simple enough to be used in everyday surroundings. In our work, we wish to provide simplicity without compromise in quality. Based on the acceptability and success of commercial motion sensor interfaces, we realize that sensor based interfaces are very easy to use in the home environment.
2.2 Data-Driven Animation using Motion Capture

Marker-based motion capture systems can generate high-fidelity animation data with subtle details of human motions. These systems perform best in the applications that mostly playback the original motions, e.g., animated movies. However, editing the original motion capture data usually results in the non-realistic animations.

There are three classes of motion editing and synthesis approaches for motion capture data: synthesis by concatenation, statistical model based synthesis and motion interpolation.

2.2.1 Synthesis by concatenation

Motion synthesis by concatenation involves motion re-ordering and re-sequencing to produce another longer motion sequence. Motion capture data can be organized into clips, then cut and combined together again to synthesize novel motions. Kovar et al. [35] introduced a graph structure called *motion graph*, to model the transition points and transition edges between different motion clips. Arikan et al. [8, 9] presented a similar graph structure with motion constraints. In a follow up work, Kovar and Gleicher [34] differentiate between logically similar and numerically similar motions and provide an automated method to identify logically similar motions from large data-sets to construct transition points. In a similar work, Arikan et al. [10] use Support Vector Machine (SVM) to annotate similar motions in a database. Their approach is logically similar poses even by using joint velocities and accelerations along with normal attributes to specify a skeleton pose. Heck et al. [29] construct a parametrized motion graph for interactive control of a character. This graph describes possible ways to generate seamless streams of motion by concatenating short motion clips generated through blending based parametric synthesis. This technique allows generation of any motion from an entire space of motions, by blending together motions from that space.
Like with Video Textures [48], such re-ordering approach cannot be used for performance-driven applications, because the subtle detail from the input signal can not be represented in the result synthesized motion by re-ordering motions from the motion capture database. However, these approaches are automated techniques and very suitable for motion estimation instead of motion capture.

The 2 major problems of motion graph based approaches are - designing a rich set of behavior for avatar, and giving the user control over these behaviors. Lee et al. [36] introduce a framework for representing, searching and producing 3D animation in real-time. The graph is a hierarchical representation for enable quick search. There are 3 interfaces for generating animations - choice based, sketch based, and a performance-driven vision based interface. In choice based interface, the user is presented with a set of possible animations which can played from the current pose. Using the sketched based interface, the user can sketch the 2D path the avatar is to follow. The vision based interface extracts visual features from video and uses them to query a motion graph. Except for the vision based interface, the other two are not intuitive enough for a simple user and are not very interactive. The vision based interface suffers from occlusion and suffers from a lag of 3 seconds. Our search algorithm shares some of the goals of this vision based interface.

In order to obtain good transition points for constructing motion graphs, many researchers [35, 8, 29] use weighted measure of similarity of two poses. However not many outline the importance of weights in selecting good transition points. Wang and Bodenheimer [54] evaluate the cost function described in [36] for determining transition points. They compute a set of optimal weights for the cost function using a constrained least squares technique. The user study showed that optimized weights produce perceptually better transition points. However, there are several limitations and possible sources of bias in their results. The optimized weights are not generic and applies only to a sample of motions. Therefore weights optimized
for a walking motion would not work well for a dynamic motion such as dancing.

The gaming industry relies on manually created graph structures called move trees [43] to represent transitions that connect several clips. A motion sequence is generated using a state machine that relies on user requests to transition from one state to another, thus playing an appropriate motion clip. Although, the process of creating move trees is labor intensive, this process is well suited for creating simple but highly interactive animations.

The single biggest advantage of using synthesis by concatenation is to provide high quality animation. We wish to take advantage of the data structures that can be generated automatically and allow matching and mixing to generate entirely new animations in real time.

### 2.2.2 Statistical modeling

Statistical models are popular candidate for synthesized novel animations that do not depend on the motion capture database. A variety of Hidden Markov Models (HMMs) have been adopted to statistically model human motions.

Brand and Hertzmann [15] use an HMM model to synthesize full-body human motions with different styles, while Bregler et al. [16] and Brand [14] use it to produce speech-driven facial animations. Li and colleagues [38] use Linear Dynamic System (LDS) model to synthesize the different full-body subtle motions. Recently, Chai and Hodgins [19] introduced a constraint-based motion synthesis system using a statistical model to address the maximum posterior (MAP) problem. Such statistical models usually require long post-processing time so they are not used in an interactive environment.
2.2.3 Motion interpolation

Motion interpolation methods allow the synthesized motion to have variations that are not presented in the motion capture database. Linear interpolation used by Guo and Roberge [28] and Wiley and Hahn [55] is straight-forward and simple. Rose et al. [46] use radial basis functions (RBFs) to interpolate motions located in an irregular parametric space. Kover and Gleicher [33] proposed a method, called “registration curves”, to identify similar motion segments in a motion database and use them to interpolate novel motions.

In a recent work, Safonova and Hodgins [47] improved motion graphs but retained the key advantages - long motion sequences, a variety of behaviors and natural transitions. They synthesize motion as a linear interpolation of two time-scaled paths through a motion graph. Interpolation provides ability to synthesize more accurate and natural motion as physically realistic variations. However, their approach is computation intensive and is therefore not suitable for real-time synthesis.

Xie et al. [56] use low-cost sensors, similar to ours, to develop a lo-cost, data-driven framework for motion estimation. This work provides the foundation for our work, since they succeed in generating high-quality data based on a local linear model learned from a high quality motion database. The linear model is an interpolation model based on RBF. Using this model and the control signals from the sensors, it is now possible to synthesize new poses. One limitation of this work is that it is an offline strategy and the synthesized result is not smooth.

While interpolation remains a popular strategy for synthesizing new poses, these approaches are computationally very expensive and real-time generation of animation would be a major challenge. Since all the poses are synthesized, smoothness must be handled carefully to give a natural quality motions. In our work, we do not face this problem because we are re-using
motion clips directly from the database.

2.3 Software architecture for 3D animation pipeline

In this section, we review the software architectures that are used for developing 3D animation pipelines.

2.3.1 Pipes and filters

Many object oriented software systems are built with modularity as the heart of design goal. The pipes and filters architectural pattern provides separation of concerns [30, 41, 31] for systems that process streams of data, for example a 3D animation pipeline. Each processing concern is encapsulated in a filter component. A performance oriented feature of this design pattern is that the filters must be able to execute concurrently since they rely only on the currently available data in the shared memory. Size of the shared memory may influence synchronization and overhead due to memory access overheads.

The idea of pipes and filters is not new. For long, Unix operating system has been using pipes and filters in the command line interface as a powerful construct that connects several independent Unix processes [52]. Another common use of this architecture is in multi-phase compilers that pipeline operations like lexical analysis, parsing, semantic analysis, optimization and code generation [24]. High performance visualization and rendering applications [58] almost always have a pipeline that processes raw stream-able data from multiple sources. A banking software may employ pipelines to enable quick integration of a module and help scale the application with frequent customer demands [42]. Recently, Yahoo! has launched a web based application called pipes [21], which is a visual programming environment for creating
a mashup RSS feed from user inputs and available RSS sources. A common theme across all of these works is that the focus has been on software architecture for ensuring modularity and code reuse, without considering system level details of the influence performance of the applications.

Assenmacher et al. [11] use pipes and filter architecture for processing streaming data in a virtual environment. Their model enables an application programmer to rearrange and create pipe elements on the fly. In their work, they identify that pipes and filters pattern is a promising technical infrastructure required to harness the power of multi-core architecture [26, 32] of hardware. In previous works, Garlan and Shaw [25] assume fairness of scheduling of filters. However, they also outline the batch processing nature of pipelines and their inability to handle interactive applications.

### 2.3.2 Layered design pattern

A layered architecture [25] is organized in a hierarchical order with each layer providing service to the layer above it and acting as client for the layer below it. The most common use of this architecture is found in network communication protocols, databases and operating systems. In contrast to pipes and filters, a layered system provide increasing levels of abstraction and allows two way communication between layers. However it is not an easy task to define the layers in a system and performance considerations may encourage tighter coupling of layers. As identified by Madsen [42] partitioning functionality across different layers often involves in compromise of the layered architecture.
2.3.3 Mixin layers

Mixin layers [49, 51, 50] is an example of collaboration based design that bridges the gap between layered design ideas and their implementations in an object-oriented language.

Archuleta et al. [7] use mixin layers to re-architecture a bioinformatics application and show how these high performance applications can be structured in a modular fashion that enables maintenance and extensibility. Their initial tests suggest that modularization using mixin layers did not compromise on performance.

Apel et al. [6] exploit mixins and design patterns for realtime and embedded systems where performance is critical on a resource-constraint environment. They treat binding mechanism as a separate concern from logic implementation. The application software can configure the software to be either compile time configurable or runtime configurable. Early binding has significant performance advantage, but less flexibility when compared to the late binding approach.

The pipes and filters pattern is the most commonly used software architecture in animation and gaming applications that involve a pipeline. The choice is based on the fact that this pattern works best for unidirectional streamable data like a continuous sequence of independent frames. For developing our application, we use the pipes and filters pattern for three reasons. First, we achieve high performance due to concurrency. Secondly, the architecture is scalable in terms of current trends of multi-core processor architecture. Finally, it is very easy to maintain and develop the entire application as a team.
Chapter 3

System Architecture and Implementation

We present a real-time motion estimation framework that tracks the user’s body movement and produces a motion that closely resembles the users action. In contrast to motion capture systems, we do not attempt to capture the subtle details of the user’s action. Our data-driven approach relies on a structured database of motion capture data and a set of sensors that are used to track body movement. Our approach consists of two phases - Data collection and representation, and real-time motion generation.

3.1 Data Collection and Representation

Figure 3.1 illustrates this step. We perform a series of off-line motion capture sessions in the studio using an optical motion capture system for animation data and a set of inertial sensors for acceleration data. The sensor data is pre-processed to reduce noise. Following this, we synchronize the motion data with the sensor data in order to get precise frame-to-frame
Figure 3.1: Data collection using Vicon motion capture system and Nintendo Wii Controllers. Ultimately, all collected data is represented as a *Motion Graph*. (*Image by author*)
mapping. All the data is then stored in a database and then converted into a structured database called *motion graph*.

A motion graph is a directed graph that connects a motion capture frame to all other similar frames such that a transition is possible while maintaining the continuity of motion. The motion graph is structured and efficient representation and allows real-time search for motion clips that satisfy a set of user supplied constraints.

### 3.2 Motion Generation

The data collection and representation are carried out only once with the final result being a motion graph. The motion graph then is used multiple times to conveniently produce different kinds of animation in everyday surroundings. Our wearable motion estimation system consists of two ordinary Wii™ controllers that transmit signals to a bluetooth capable terminal. During motion generation, these input sensor signals coming from different sensors are synchronized in time to each other and any noise is removed. We then use the input signals to query the motion graph for walks in the graph that match to the input signals. The walk consists of motion clips that are stitched together and blended at the transition points to create a continuous motion sequence. The motion estimation system architecture is shown in Figure 3.2

### 3.3 Application software design and implementation

Our software application is a real-time 3D animation pipeline with a modular design and data stream. The *pipes and filters* architectural pattern is best suited to implement this design. Such data pipelines are popular in computer graphics applications where a series
of transformations are performed on a stream of data. The pipes and filters architectural pattern provides benefits of uniform interconnection methods to construct a chain of independent processing entities. Each processing entity is called a filter. A pipe interconnects adjacent filters and is responsible for synchronized data transfer between the filters. Pipes can implement communication using shared memory or distributed memory. While distributed memory would provide the flexibility to run filters on separate machines, the shared memory would enable faster pipes since data movement is restricted to the same physical memory. Pipes are attached to filters at ports. The pipes and filters architecture can be configured to support multiple combination of filters and in flexible order in the pipeline.

Data originates from a source and is finally delivered to a sink that resides at the end of the pipeline. The generic pipes and filter architecture is shown in Figure 3.3 (a).

For our application, the source for the pipeline is data from the 2 accelerometer sensors which together produce a single packet (also known as frame). A packet is unit of transmission in the pipe. The data packets from the sensors pass though several filters and finally used as control signal to drive animation of a human avatar. The sink for the pipeline is a rendering engine. The 3D animation program was implemented in C++ and supports two variations for filters - active and passive. The application with active filters was implemented using POSIX [17] compliant pthreads.
A filter may be *active* or *passive*. An active filter runs as a separate thread and requires the service of pipes to synchronize data transfer between adjacent filters. This type of filter actively pulls data from the input pipe and after applying a transformation, pushes it immediately onto the output pipe. On the other hand, a passive filter is activated by a series of procedure calls initiated by the source. Alternatively, it could be activated by a series of procedure calls originating from the sink. For our application, we chose active filters to provide concurrency for improved performance on multi-core hardware. In fact each every processing unit is an active filter, running on a separate core of a multi-core cpu.

Since filters do not share any state, each filter can be developed by separate teams. Therefore pipes and filters pattern inherently supports maintainability because filters are independent entities and can be interchanged, replaced or reconfigured. For example, let us consider a case where we would like to add a new module to an existing pipeline as shown in Figure 3.3 that process the stream of packets to remove noise in the data. First we must implement a class say, \texttt{NoiseReductionFilter} that derives from class \texttt{Filter}. Next we create a new \texttt{Pipe} called \texttt{pipe3}, and connect as follows using the following 3 lines of code.
Pipe *pipe3 = new Pipe();
Filter *noiseReductionFilter = new NoiseReductionFilter(1, pipe2, pipe3);
noiseReductionFilter->run();

We must also reconnect the output and input pipes for Filter2 and Filter3, respectively. Finally, class NoiseReductionFilter must implement just one method called the applyTransform(Packet * pkt), where the noise reduction code is added to process incoming packet pkt. Figure 3.3 (b) shows the new pipeline after connections have been successfully made.
Chapter 4

Data Collection and Representation

The quality and size of captured data is crucial for obtaining good results with our framework. High quality motion capture data will ensure high quality animation and good set of transition points in the motion graph. An appropriate size for the database will help build an efficient motion graph for real-time search operations. In this chapter we will first discuss the initial steps that contribute towards building a motion graph. We then describe how we represent motion data in the form of a motion graph.

4.1 Data Collection

4.1.1 Data Capture

During the studio motion capture session, we capture 2 types of data. The first is a high quality motion capture data, captured using Vicon optical motion capture system. Synchronously, we capture acceleration data using accelerometer sensors attached to the limbs of the performer. Only one motion capture session was required in which we captured a
total of 86 seconds of data as shown in Table 4.1. For building the database used for the examples presented in this paper, we had only one subject perform all the actions.

<table>
<thead>
<tr>
<th>Action</th>
<th>Number of frames</th>
<th>Duration (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arm exercise</td>
<td>818</td>
<td>13.63</td>
</tr>
<tr>
<td>Tennis</td>
<td>1929</td>
<td>32.15</td>
</tr>
<tr>
<td>Basketball</td>
<td>816</td>
<td>13.60</td>
</tr>
<tr>
<td>Golf</td>
<td>1597</td>
<td>26.62</td>
</tr>
</tbody>
</table>

Table 4.1: Composition of motion database.

For optical motion capture, we use a Vicon system with 8 Vicon MX series cameras for high quality motion capture at 60 Hz. The motion sensors are 3D accelerometers (Wii™ Nintendo controllers) with a range of ±3g and built-in Bluetooth® interface for data transmission at a peak rate of 100 Hz. The interface based on these wireless inertial sensors is cheap, easy to set-up and unlike vision based system, does not suffer from occlusion.

Figure 4.1 shows a total of 45 retro-reflective markers and eight sensors attached to the performer’s body. The sensors are attached to the arms and legs since they provide most of the movements for a majority of human actions. For our database shown in Table 4.1, we only used data from the 2 sensors attached to each forearm. Actions present in our database mostly employ movement of the arms and therefore the 2 sensors provided sufficient constraints for the search algorithm. For complex actions involving legs, we would be required to use more sensors. Each sensor transmits its 3D acceleration data to the data collection computer where the data is passed through a noise reduction filter. Next, it is converted into sensor frames, synchronized with motion data frames and stored into the database.

The created database with $N$ frames of data is a collection of values of the form $(c_t, m_t)|t = 1, \ldots, N$ where

$$m_t = F(c_t)$$  (4.1)
Figure 4.1: Data collection: an optical motion capture system and a 3D acceleration sensor based data acquisition system are used in parallel. There are 45 retro-reflective markers and eight sensors (four Wii™ Nintendo controllers) attached to the performer. (Image by author)
\(c_t\) is a frame of sensor data with 6 dimensions and it represents the 3D acceleration measures of four or eight sensors on the body at time \(t\). \(m_t\) is a frame of optical motion capture data and it represents a pose at time \(t\). The pose is represented in the form of local joint rotation in the quaternion format. \(F\) is an injective function that uniquely maps each sensor frame to the pose at time \(t\).

### 4.1.2 Noise reduction filter

The Wii controllers use Bluetooth Human Interface Device (HID) [45] protocol for data transmission between the controller and the terminal. Due to unreliable transmission in a wireless medium and a best-effort Quality of Service, data packets are often lost and sometimes even get corrupt. We refer to these corrupt values as noise. We deal with lost packets in Section 4.1.3.

If \(c_t\) is the sensor frame received, it is detected as noise if any of it’s 3-dimensional component is not in the range of expected values for the device which is \((0 - 255)\). The corrupt frame is replaced by an estimate which is the average value calculated from the frames in its neighborhood. Thus for a neighborhood of size \(2r\), the estimated frame \(c'_t\) is given by

\[
c'_t = \frac{\sum_{i=-r}^{+r} (c_{t-i})}{2r}
\]  

(4.2)

We chose \(2r\) to be 5, based on the observation that consecutive corrupt frames do not occur and \(c_t\) is a smooth curve.
4.1.3 Data Synchronization

After motion capture, we must map each motion data frame $m_t$ to its corresponding sensor data frame $c_t$. But first we have to construct a sensor frame $c_t$ by combining acceleration data from each of the sensors which have independent channels for transmission. Additionally, data from each sensor is received with variable frame-rate due to varying conditions in the wireless environment.

To resolve these two issues, all data received from sensors are marked with a sensor identifier and placed in a common buffer. A snapshot of the buffer is taken after every frame period and one sensor frame is constructed with the data received in the current interval. After a snapshot is taken, the buffer is overwritten if new data arrives. Typically, there should be only one sensor frame in the buffer when taking snapshots. However, if the terminal failed to receive data from any sensor during this time period (or the buffer is empty), then the previously received frame is considered again. If there are more than one sensor frames, we use the average of all frames in the buffer. This way the sensors can be synchronized and a constant frame-rate of 60Hz achieved to match the frame rate of the motion capture system. We also use this strategy to synchronize the sensors and obtain constant frame rate during the motion graph search phase.

After the sensors are synchronized with each other, the next step is to synchronize the motion capture data with the sensor data. For this we asked the performer to strike his fists before and after performing any action. We can use this striking event to synchronize the sensor data and motion capture data, by aligning the spike produced in the sensor readings with the frame in motion capture data when fists are closest.
4.2 Data Representation

Given a database of motion capture data, we create a directed graph called a *motion graph*. The idea of motion graph is not new and the structure of our motion graph that we use is largely inspired by the data structure of Kovar et al. [35]. There are two kinds of edges in the graph. As seen in Figure 4.2, the solid edge represents a piece of original motion capture data which we will refer to as *clip*. The dashed edge represents a *transition* from one end of a clip to the beginning of another. Thus the nodes are junctions called *transition points* where transitions begin and end. Transition points are also the place where a clip starts or ends. A local loop is a transition to self node. Self loop has been introduced for uniformity and is required to transition to the outgoing clip edge from current node. Say if we are at node 5 and we need to transition from this node. One of the possible transitions should allow us to play the clip between nodes 5 and 6. Thus if we transition from node 5 to itself, we must play the clip that begins from 5.

In Figure 4.2, node 7 is a dead-end node because there is no outgoing edge from this node. Node 6 is a weakly connected node since it is reachable only to a limited part of the motion graph though the only possible transition. The motion graph is built only once and stored in a text file. In every subsequent run, of the application, the motion graph is loaded from this file at run time.

4.2.1 Identifying Transitions

We represent the unstructured motion capture database built in Section 4.1, as a collection of motion sequences $M_1, M_2, ... M_n$. A transition from $M_x$ to another motion sequence $M_y$ consists pair of frames $A_i$ in $M_x$ and $B_j$ in $M_y$ that are similar enough so that we can stop playing $M_x$ at frame $A_i$ and begin playing frame $M_y$ from $B_j$, without any jitter. The
Figure 4.2: An example motion graph generated from two sets of motion capture data

transition edge connects a node at \(i^{th}\) frame in \(M_x\) and another node at \(j^{th}\) frame in \(M_y\). When motion is played along a path in the motion graph involving 2 clip edges connected using a transition edge, the result is a new motion sequence.

In order to generate the complete motion graph, our first task is to identify all the candidate transition points. Given two motion sequences of size \(S_x\) and \(S_y\), we calculate a distance matrix \(D\) of size \(S_x \times S_y\), such that \(D(i,j)\) is a measure of similarity of \(i^{th}\) frame in motion sequence \(M_x\) and \(j^{th}\) frame in \(M_y\). This method is also used by Kovar et al. [35] and Schdl et al. [48]. \(D(i,j)\) is the weighted sum of squared distance between the frames and is given by

\[
D(i, j) = \sum_{k=0}^{TJ} w_k ||q^i_k - q^j_k||^2
\]  

(4.3)

Here \(q^i_k\) and \(q^j_k\) are the \(k^{th}\) joint angle of the skeleton for \(i^{th}\) and \(j^{th}\) poses in motion sequences \(M_x\) and \(M_y\) respectively. \(TJ\) is the total number of joints in the skeleton and \(w_k\) is weight associated with joint \(k\). Lee et al. [36] used binary weights with values of 1 for important joints and 0 for others. An important joint is one whose change in value brings a significant change in the pose. We apply a similar strategy and apply weights of 1 for spine, upper-leg,
knee, shoulder and elbow. The global translation and orientation is ignored in all calculations because a pose is defined only by the angles in the skeleton’s coordinate system.

There are some poses that are numerically similar but are logically different. For example two poses may be exactly similar but one is part of a motion when a user is lifting his arm and the other pose will occur when the user brings down his arm in the reverse direction. Transition between these poses should not be allowed because they are part of logically different actions. To avoid this we should incorporate angular velocity and acceleration of joints. We can calculate angular velocity from from 2 adjacent frames and acceleration from 3 consecutive frames. Instead of calculating the velocities and acceleration, we can incorporate these higher-order derivatives by taking a window of frames of size WND for calculating individual distances. In our case we have set WND to 3. Thus, \( D(i, j) \) is given by

\[
D(i, j) = \sum_{l=0}^{\text{WND}} \sum_k w_k \| q_{k-l}^j - q_{k-l}^i \|^2 
\]  \hspace{1cm} (4.4)

An example distance matrix is shown in Figure 4.3 for two sample motion sequences. The white portions indicate high similarity while darker region means the opposite.

The probable transition points are located in the white regions. It is common to find a whole neighborhood of frames having good and very similar transitions to some other neighborhood of frames. However, we only need the best transition amongst these candidates. We use convolution to find the local minima and select the sweet spots for transitions. These sweet spots or candidate transition points are indicated in Figure 4.3 as red spots. The final transition points will be selected after we have pruned some bad transitions. It must be noted that given any two frames, they would either have a transition between them or not at all - we leave the task of finding the quality of a transition for the search phase.
Figure 4.3: The distance matrix for two sample clips - the red spots have been identified as transition points

4.2.2 Pruning the Motion Graph

Pruning the motion graph for unwanted transitions is required for the following reasons.

- remove acyclic regions in the motion graph
- improve quality of resulting motion sequence by suppressing non-optimal transitions
- reduce complexity and save on storage by having a compact motion graph

Pruning Based on Distance Threshold

After obtaining the local minima we prune the transitions to accept only those that are above an acceptable quality. The acceptable level of quality is defined by the distance threshold. We reject those transitions between frames that have a distance measure (using Equation 4.4) that is above this threshold. The threshold value is not global and it can be set to a different value for every pair of motion sequences to control the quantity of transitions. Pruning based on distance threshold is largely responsible for controlling the overall quality and total number of transitions.
Merging Similar Transitions

Even though we apply convolution to obtain the local minima for selecting transitions, there will be transitions that have source frames which are adjacent in the motion sequence and very identical to each other while the destination frames are scattered. We must detect these nodes and represent them with one single node while keeping all the transitions. Our compression technique retains the functionality of the motion graph with the help of a more compressed and efficient transition. Figure 4.4 (a) shows a simplified version of a motion graph with 7 nodes and 3 transitions. Nodes 2, 3, and 4 are neighboring nodes, only a few frames apart and each with transition to nodes that are not neighbors. Note, that convolution will not allow multiple transitions from one neighborhood to another, but it will allow transitions as shown in Figure 4.4 (a). Merging the source nodes results in the motion graph shown in Figure 4.4 (b). Since nodes 2, 3 and 4 have logically similar start frames for transition, we have removed nodes 2 and 4 and they are all now represented by node 3. After this merge, it is now possible to transition from node 3 to many more places in the motion graph, thus providing higher connectivity to the entire graph. As shown in Section 5, this scheme will be beneficial in the search phase of the motion graph. Similar to scenario described above, we will have another scenario in which the source nodes for transitions are
not in a neighborhood, but the destination nodes lie in the same neighborhood. Although merging these nodes have no effect as far as the search algorithm is concerned, it will help in making the motion graph more compact, thus reducing memory consumption and boosting performance.

**Removing Dead-ends**

A dead end in the motion graph occurs when there is a node which after which no transition is possible. Dead ends are not part of any cycle and the produced motion will come to a halt on encountering such a node. Any node that does not have any out-going edge is a dead-end node and is removed from the graph. Removing a dead-end node may cause its adjacent node to become a dead-end if the connecting edge between the two was the only outgoing edge. We iterate thought this process and remove dead-ends until none exist.
Chapter 5

Searching the motion graph

In the previous chapter we built a motion graph from the motion capture database. In this chapter, we describe the process of generating motion sequences by searching the motion graph for motions corresponding to actions being performed by the user. The user wears two sensor controllers in each of his arms. First we establish network connectivity with the computer where animation is to be rendered in real-time. The total setup time is less that 2 minutes. Since our system does not suffer from occlusion, the user can position himself in any direction, but should be close enough to the terminal to maintain Bluetooth connectivity. For examples in this work, the user performs several actions that mostly require movement of the upper body. For this reason, it is sufficient to use only 2 controllers.

At this stage there are 2 tasks to accomplish. First we must search in the motion graph for paths that satisfy user constraints in real-time. Secondly the resulting graph walk needs to be converted to continuous motion.
5.1 Local online search

Given a current node, our novel graph search algorithm uses the input sensor signals as the query for searching a matching transition from the node. Our approach is local because we only search for transitions possible from the current node. Secondly, our approach is online because the results are obtained and rendering is performed at real-time rates. Unlike the graph building phase, in the search phase we will work only with motion sensor frames. Let us recall that before creating the motion graph we built a database of motion capture data with one-to-one correspondence with motion sensor data. These one-to-one correspondence is maintained even in the motion graph. Thus, every motion capture frame of a clip edge in the motion graph has a corresponding motion sensor frame.

At any given time $t$, we maintain a sliding window buffer, $U_t$ of size $CSW$ for the input sensor signals. If the current node is $n_t$, we obtain all the transitions possible from $n_t$. Then we find the minimum of the distance between the input signal and each of these transitions as the weighted square of Euclidian distance.

$$d_{\text{min}} = \min\left( \sum_{i=0}^{CSW} w_i ||cs_i - cs'_i||^2 \right)$$  \hspace{1cm} (5.1)

Here $cs_i$ is an input sensor signal frame in the buffer $U_t$ and $cs'_i$ is a sensor signal frame from the motion graph corresponding to a transition from $n_t$ with $cs_0$ being the first frame in the destination clip edge. The weights $w_i$ is a linear weight given by

$$w_i = \frac{1}{CSW}$$  \hspace{1cm} (5.2)

Let us assume that the first and the last frame of the current clip edge is $A_{i-l}$ and $A_i$
respectively where \( l \) is the length of the clip edge. The process for finding \( d_{\text{min}} \) begins when \( A_{i-l} \) is being rendered and is repeated after every subsequent frame until we reach frame \( A_{i-\frac{k}{2}} \). The choice for this frame is based on our blending technique and is discussed in Section 5.2.1. After every iteration we update \( d_{\text{min}} \) if a lower value is found. When we reach the frame \( A_{i-\frac{k}{2}} \), we stop searching and proceed with next step - to select an appropriate transition based on our search results. At this state our search algorithm has yielded \( d_{\text{min}} \), which is the transition to the clip which is closest to the query signal.

If \( d_{\text{min}} \) is below an acceptable distance threshold, we select the transition corresponding to it and proceed with blending as described in Section 5.2.1. However, it would be easy to encounter situations in which \( d_{\text{min}} \) is above the distance threshold and the transition is acceptable. This would typically happen if the user is performing an action that does not exist in the database. It would also happen if the user did not perform an action well enough to get recognized. In such cases we select a transition to a node in a special sub-graph called \textit{default motion sub-graph}. However, not all nodes will have a transition to the default sub-graph. In this case we choose a self-transition and continue playing frames from the new clip. We continue playing clips in this fashion until we either find a matching transition through new search or we encounter a transition to the default motion sub-graph.

### 5.1.1 Default motion sub-graph

A default motion sub-graph is our extension to a pure motion graph based approach. This is a special graph constructed from 30 frames and every frame is a node in the graph. There is only one gateway node for all incoming transitions. However, all nodes have outgoing transitions. The outgoing transitions are same for all the nodes. The default motion sub-graph is constructed from a specially selected motion capture data in which the skeleton
pose is in a neutral position in all the frames. In order to find the default motion sub-graph for a clip, we find a pose that occurs most often in the clip. We use the same distance metric as shown in Equation 4.4. This pose is acts like a hub pose for all actions. Figure 5.1 shows 2 default poses that we use for the examples in our database. When in default pose, the avatar waits for a user to perform an action.

Figure 5.1: Default pose - an arm lifting action is more likely to transition to (a), golf to pose shown in (b) and tennis, basketball to (c)

Once the default poses are found, we take 0.5 second (30 frames) of continuos motion from a clip such that all the frames are very similar to the default pose. We will call this motion sequence as the *default motion sequence*. The default motion sequence can be easily found by running though the entire motion capture database and finding a sub-sequence that is 60 frames long and all frames are very similar to each other. Using these frames, we construct a small and special motion graph called the *default sub-motion graph* which has as many nodes as number of frames in the default motion sequence. The end node is connected to the beginning node, forming a circle of continuous motion. We now connect this sub-graph to the
rest of the motion graph by creating bi-directional transitions from all clip edges whose end
frame is similar to the default frame. Finally, it should be possible for us to transition from
any node in the default motion sub-graph to the selected nodes of the motion graph without
waiting for the entire default clip to finish playing. For this we duplicate all the transitions
that were created for the first node of the default motion sub-graph, to the selected nodes
of the motion graph.

The default motion sub-graph is a unique feature and is required to play animation whenever
the search algorithm returns a poor transition. It is also used to play a clip that indicates
that the system is waiting for user to perform an action.
5.2 Generating motion

At this stage we have identified a path consisting of clip and transition edges in the motion graph that should be converted into continuous and smooth motion. Since we are stitching clips together, we must first align the clips such that the start frame of new clip matches the global translation and orientation of the previous clip played. Then, we must blend the motion at the junction to produce smooth transition.

5.2.1 Blending

After selecting a transition, we perform blending so that the switching from one clip to another is smooth. Given two clips $M_x$ and $M_y$, we wish to transition from the last frame $A_i$ of $M_x$ to first frame of $B_j$ of $M_y$. To achieve this we blend frames $A_{i-k/2}$ to $A_{i+k/2}$ with frames $B_{j-k/2}$ to $B_{j+k/2}$. The first step is to align the clip $M_x$ with $M_y$ as described in the previous section. Then we perform spherical linear interpolation on the joints rotations.

\[
q^{AB}_p = \text{Slerp}(\alpha(p), q^{A}_{i+k/2-p-1}, q^{B}_{j+k/2-p-1}) \tag{5.3}
\]

$q^{AB}_p$ is the resultant quaternion of the $p^{th}$ blend frame created by combining respective quaternions from frames $A_{i+k/2-p-1}$ and $B_{j+k/2-p-1}$. $\alpha(p)$ is a blend weight chosen to enable a smooth blend and is given by

\[
\alpha(p) = 2\left(\frac{p+1}{k}\right) - 3\left(\frac{p+1}{k}\right)^2 + 1; 0 < p < k \tag{5.4}
\]

We have chosen $k$, the blend interval to be a dynamic value with a maximum value of 20 frames, or equivalent of 0.33 seconds. The value of $k$, depends upon when the transition
begins with respect to the length of the clip edges. The blending scheme is given in Figure 5.3

Figure 5.3: Creating transition from node \( i \) to node \( j \) by blending frames in a window

In Section 5.1 we discussed about searching for a suitable transition while a clip edge is being rendered. We also mentioned that the search concludes when we have reached frame \( A_{i-k/2} \), where \( k \) is the blend interval. For obtaining best results from blending, we decided to have \( A_i \) and \( B_j \), the frames being blended, at the center of the blend window. Keeping these frames at the center ensures that the blended frame has fair contribution from frames from each clip being blended. There may be situations such that the next transition may happen before the previous has completed. To handle this, we maintain a buffer of frames generated from the previous transition, but not used. We will use these frames for generating the next transition instead of original motion frames for the overlap between the current and previous blend intervals.
Chapter 6

Results

Our system is capable of capturing all motions that are part of the motion database. The examples presented in this thesis were run on a terminal with system configuration as summarized in Table 6.1. The real-time frame rate achieved 60 frames per second. The user has two Wiim™ controllers attached to his arms and connected to the terminal via Bluetooth®. The controllers are oriented in the same direction as they were attached to the arms of the professional performer during data collection phase. We asked the user to perform tennis forehand, tennis serve, basketball dribble, basketball shot, arms exercise and golf swing. The user performed these actions in random order. A sequence of skeleton poses along the corresponding images from video is also shown in Figure A.2. The quality and accuracy of animation produced is shown in the accompanying video.

6.1 Evaluation

For evaluating our system, we follow a two step procedure. In the first step we measure the accuracy of our search algorithm to correctly match user actions with motions in the
Table 6.1: System Configuration

<table>
<thead>
<tr>
<th>Processor</th>
<th>2.33 GHz Intel Core 2 Duo</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2 Cache</td>
<td>4 MB</td>
</tr>
<tr>
<td>RAM</td>
<td>2 GB</td>
</tr>
<tr>
<td>Graphics card</td>
<td>NVIDIA GeForce 7600 GT</td>
</tr>
<tr>
<td>Operating System</td>
<td>Mac OS X</td>
</tr>
</tbody>
</table>

database. In the second step, we measure the stability of our system.

6.1.1 Accuracy

In this step we will measure the accuracy of our approach. This measure reflects the ability of our search algorithm to find best match for user actions. If the user performed an action $N$ times, we measure the number of times the action was correctly recognized ($CR$), number of times the action was incorrectly recognized ($IR$) and number of times the action was not recognized ($NR$). An action is considered not recognized if the avatar does not respond to the user action and instead maintains the default pose. Each of these measurements are summarized in Table 6.2 for $N = 50$, the number of repetitions of each action.

Table 6.2: Accuracy of the online, local search algorithm

<table>
<thead>
<tr>
<th>Action</th>
<th>CR/N</th>
<th>IR/N</th>
<th>NR/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Right arm dumbell</td>
<td>0.88</td>
<td>0.00</td>
<td>0.12</td>
</tr>
<tr>
<td>Left arm dumbell</td>
<td>0.84</td>
<td>0.0</td>
<td>0.16</td>
</tr>
<tr>
<td>Right arm lift</td>
<td>0.96</td>
<td>0.0</td>
<td>0.04</td>
</tr>
<tr>
<td>Left arm lift</td>
<td>0.96</td>
<td>0.04</td>
<td>0.0</td>
</tr>
<tr>
<td>Tennis forehand</td>
<td>0.82</td>
<td>0.0</td>
<td>0.18</td>
</tr>
<tr>
<td>Basketball dribble</td>
<td>0.88</td>
<td>0.06</td>
<td>0.06</td>
</tr>
<tr>
<td>Basketball shot</td>
<td>0.76</td>
<td>0.0</td>
<td>0.24</td>
</tr>
<tr>
<td>Golf</td>
<td>0.94</td>
<td>0.0</td>
<td>0.06</td>
</tr>
</tbody>
</table>
These results indicate that we have achieved about 91% accuracy for short actions like arm exercises and basketball dribble. The best results is for arms exercise and the worst is for basketball shot. The accuracy of recognition is dependent on the amount of time taken for preparation of an action. If is also dependent on the amount of variance in the sensor signal being received and the sensor signals in the database. The overall accuracy is 87.0%.

### 6.1.2 Stability

The stability test will tell us if our system is intelligent enough to recognize intervals when the user is not performing any action. During such cases, it is expected that an appropriate default clip is being played giving the impression that the avatar is waiting for the next instruction from user. An error condition occurs if the system wrongly plays a motion sequence, instead of remaining in default state. Let us call this a *Type I* error. For measuring the Type I error, we asked the user to stand in each of the three defaults poses.

In the second case, if the user performs arbitrary actions (not present in database), then the search algorithm should not return anything and an appropriate default clip is played. The action to be performed for this test is chosen carefully so that it is not similar to any of the actions in the database. Again, an error condition is indicated if the system wrongly plays a motion sequence, instead of remaining in default state. We call such errors as *Type II* errors.

For this test, the user was asked to perform two actions. First we asked the user to move right arm in circular fashion, as if drawing a circle on a blackboard. The second action was to move both arms as if the user is running. Both of these actions are not present in the motion database.

All actions were performed continuously for 30 seconds and each action was repeated 10 times. The results are presented in Table 6.3. We see that while the user is not moving,
there are no errors, but when the user is moving arbitrarily, the system is occasionally confused and selects a wrong transition. However, the low percentage in error for Type II errors suggests that our system is stable during such conditions.

Table 6.3: Stability test

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Action</th>
<th>Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type I</td>
<td>Default pose 1</td>
<td>0</td>
</tr>
<tr>
<td>Type I</td>
<td>Default pose 2</td>
<td>0</td>
</tr>
<tr>
<td>Type I</td>
<td>Default pose 3</td>
<td>0</td>
</tr>
<tr>
<td>Type II</td>
<td>Running</td>
<td>0</td>
</tr>
<tr>
<td>Type II</td>
<td>Right arm circle</td>
<td>10</td>
</tr>
<tr>
<td>Type II</td>
<td>Dumbbell, both arms</td>
<td>0</td>
</tr>
<tr>
<td>Type II</td>
<td>Arm lift, both arms</td>
<td>0</td>
</tr>
</tbody>
</table>

6.2 Applications

Our goal is to make motion capture as easy as video capture so that it is used by a simple user in everyday environment. We would like to make it applicable for a wide range of interactive applications, from video game interfaces, animated chat-rooms to character control in virtual environments. Our target applications would like to use motion data with emphasis on quality and natural motion, while still providing interactivity.

6.2.1 Interactive character control

Using just two controllers, we can control an animated character at real-time speeds. This kind of interactivity is needed in video games and virtual environments. The Wii™ Controller is already being used for this purpose in many games. However none of those games provide precise control over the character and the animation lacks the quality of motion capture
data. Our system provides a Wii™ controller based interface which is very intuitive to use. The user moves his body in the same fashion as he would like the character to move, thus reducing any learning curve.

6.2.2 Motion capture

While our framework is not suitable for a motion capture in true sense, it can be used to generate motion data of the same quality as any studio motion capture data. Thus our framework is a motion estimation framework since we aim to generate logically similar motion data, rather than numerically similar data. The captured motion can be saved to the disk and used later for movie and video game applications.

6.2.3 Random graph walks

A common application of any motion graph based approach is a random graph walk resulting in random but smooth animations being generated. The motion graph is a cyclic graph, that can provide random graph walks without exhausting. This random walk can then be used to generate seamless, high-quality motion streams in realtime. Random motion streams are widely used in animation movies and video games for background characters and for generating random crowd.

To generate random graph walk we being from any node and find the number of transition edges possible from that node. We then randomly choose a transition edge and select the following clip for generating motion stream.
6.3 Discussions

The results presented show that motions for interactive characters can be generated by concatenation and blending of smaller motion clips. Our method provides an intuitive user interface for animating the characters. We will now analyze some of our findings that affect search results and how we handled them. We will then discuss how the software architecture we implemented helps attain real-time performance. We will conclude this section with a few limitations of our system.

6.3.1 Search algorithm performance

The online search algorithm is very efficient because it only searches the best matching transition to a neighbor node. This local approach combined with a concurrent software design allows us to achieve real-time speeds without compromising quality. However, we face 2 problems while performing the search. The first problem arises when there are two actions that result in similar sensor values, while the second problem is related to lag time. Finally, we discuss how the structure of the motion graph relates to performance.

Similar sensor readings

We are unable to differentiate between two different actions that have very similar 3D acceleration values. For example tennis forehand and tennis backhand. In Figure 6.1, we see the readings from the right arm sensor. On the left side we see the plot for two tennis forehand strokes and on the right for backhand strokes. These plots are visually and numerically very identical. The readings from the left arm sensor also show similar behavior. In this case, our search algorithm will select either of the two actions, resulting in wrong actions being
generate. We could solve this problem by increasing the search window $CSW$, but this will result in increased lag. We therefore eliminated backhand from our database.

![Figure 6.1: Sensor readings from the right arm sensor for tennis forehand (left) and tennis backhand (right)](image)

**Lag**

In our approach we are trying to match user actions in the past with skeleton actions in the database that will be played in future. This means that there will always be a lag. In our case the lag is $\frac{CSW}{FPS}$ or 0.83 seconds for a $CSW$ of 50 frames while maintaining 60 frames per second.

In Figure 6.2, we have shown the readings of sensor values for different actions for a window of 80 frames. Figures (b), (d) and (e) show the a smaller window of size 50 frames with a dashed line. The size of search window $CSW$, is set based on the variance of the received signals measured in the window interval. If the variance is high, we know that signals have a rapidly changing pattern which is good for recognition. We get a sufficiently high variance for arm lifting actions for a window size of 50. If the variance is low, as in the case of Figure 6.2 (e), then we must use a larger window size so that we are not trying to compare with the 'flat' portion of the sensor data. This flat portion is because some actions like tennis and golf have a preparation time before the actual motion characteristic to the action begins.
Figure 6.2: Sensor readings for (a) left arm dumbbell lift, (b) right arm dumbbell lift, (c) left arm lift, (d) right arm lift and (e) tennis forehand. The top row shows readings from left sensor while the bottom row shows readings from right sensor.

We must therefore have a larger $CSW$ of 80 frames for such actions. However, increasing the $CSW$ to 80 increases the lag to 1.33 seconds, something that is not desired.

### 6.3.2 Software Design for Real-time application

Our application is designed using the pipes and filters software architecture. This design was necessary to get the the best performance to support real-time processing of frames. The choice of architecture was natural because we have a pipeline though which frames pass and they are processed at each stage independently. The boost in performance comes from the fact that each processing module (filter) is running in parallel and the user’s terminal has multi-core hardware architecture. Additionally, we ensure that there is no bottleneck due to insufficient shared memory between two filters. Using such an architecture, we were able to support a frame rate of 60 frames per second, which is not common for current real-time applications.
6.3.3 Limitations

The type of animation that can be generated using our system is only limited by the size of the database and sensitivity of the sensors. Our system suffers from the basic limitation of any motion graph based approach - the motion sequences being generated must be present in the database. The motion graph only helps to concatenate several smaller clips to yield a larger continuous motion sequence. This means that if a new motion is required, then a new motion graph must be built after capturing that action in the studio. With sufficient pre-planning, we can easily avoid this scenario.

A second limitation of our approach is that we are unable to avoid lag. As explained in Section 6.3.1, we are trying to match user actions in the past with avatar actions in the database that will be played in future. Therefore our system is most suited for actions that occur quickly. For example punching and kicking. However, with more sensitive sensors and a dynamic control signal window (CSW), we can attempt to reduce the lag, if not eliminate it completely.
Chapter 7

Conclusions and Future Work

7.1 Conclusions

We present a framework for estimating full body motion in real-time using a small set of low cost inertial sensors. Our three step approach involves data collection, building a motion graph and motion generation though local graph search. Data collection is performed in the studio and produces a database of time synchronized high quality motion capture data and sensor data. Using this database, we then construct a commonly used data structure for 3D animations called a motion graph. Creating an efficient motion graph is not trivial, since it involves finding the best transition points from a pool of candidate transitions. We prune the graph to remove redundant transitions and dead-ends and introduce a new compression technique to improve search performance.

Using this motion graph, we can generate various new motion sequences by concatenation of clips obtained from the motion graph search. In the search phase, we proposed a local online algorithm that uses the sensor signals as a query to the motion graph and returns an
appropriate transition corresponding to the user’s current action. When the search algorithm
is unable to understand the user’s action we play the default motion clip until the user
performs an action that can be recognized. We extended the commonly used motion graph
technique to introduce a default motion sub-graph. A walk through this circular sub-graph
produces the default motion clip.

The results obtained show the effectiveness of our framework. We achieved accuracy
of 87.0% while maintaining a lag of 1.33 seconds. Using a sufficient number of examples
we can synthesize a large variety of human motions. The quality of the generated motion
sequence is same as original motion capture data, since we follow a cut and paste strategy
and synthesize frames only during transitions between different clips. The accuracy and lag
may not be suitable for applications that require immediate response. Thus, in our future
work, we wish to increase the accuracy and reduce the lag so that our framework can be
used in a wide variety of applications.

We chose the pipes and filters architecture for software design which allows development of
maintainable code and boosts performance to highest levels though concurrency. This design
contributes significantly towards developing an interactive application. Due to this software
architecture and a local search strategy, our framework performs with real-time speeds and
maintains 60 frames per second output. Additionally, the architecture allowed development
of maintainable source code that is very flexible for the development of a pipeline.

7.2 Future Work

There are several improvements we plan to do in the near future. Our sensors do not produce
high quality results for very fast actions like karate punching and kicking. We can overcome
these problems by using sensors with sensitivity of ±10g and increasing the frame rate to
120Hz for the optical motion capture system.

We would like to increase the size of the database by incorporating clips with locomotion. Like most of the statistical model based motion synthesis, when we use such clips, our approach suffers from foot sliding problem during transition. We plan to address it by using constraint based motion editing techniques [27, 37].

The current lag is around 1.33 seconds. This may be too large for some application that require high responsiveness. Using our approach, it is not be possible to reduce the lag completely but more research is needed to shorten $CSW$, and still get accurate results. One approach would be to predict the action based on a much smaller $CSW$ and start playing the clip. We then search for the action based on a larger $CSW$. If our prediction was correct, we continue playing the clip else we switch midway and maintain smoothness using blending.

Finally, we would like to improve our user interface based on Wii™ controllers. We would like to user sensors that are wearable in a real sense. This can be achieved by using taking off the essential devices from within the Wii™ controller and stitching them to wrist bands. An alternate interface would be using the Wii™ controllers but without caring for its orientation in the users hand. Thus, the user should be able to simply hold it and obtain the equivalent results.
Bibliography


Appendix A

Generated motion clips

Figure A.1: Software implementation consists of modified version of open-source Darwin Remote, 2 pipelines for data collection and online search and a 3D animation renderer
Figure A.2: Five different actions (one in each row) generated by our system. Each frame shows on the left side the actual pose and on the right side the a pose from the generate motion sequence. For the purpose of comparison, we have presented the results after removing the lag. (Image by author)