Motion Graphs

Christian de Chenu
B.A. Mathematics
Final Year Project 2006
Supervisor: Dr. Steven Collins

Abstract

With the increasing use of motion capture in video games, research has been devoted to methods of controlling motion capture data. Given a set of motion capture data, we automatically construct a directed graph called a *Motion Graph* which allows the editing and combining of segments of captured motion. This project aims to research, implement and evaluate a system for displaying motion capture data which includes the Motion Graphs concept.

Acknowledgements

Steven Collins, my supervisor, firstly for his support and help during the research, and for offering the project in the first place. Also Simon Dobbyn for lending his time to show me the motion capture system in action.

Contents

Chapter 1

Introduction

1.1 Overview - Motion Capture

"Motion Capture is a technique for recording the movements of a body in motion and translating them into usable mathematical terms by tracking a number of key points in space over time and combining them to obtain a single three-dimensional representation of the performance", [ME99]. The captured body could be anything that moves. Today's movies and video games require animated characters to move realistically within their world. This is where the idea of motion capture comes in. The process involves the person wearing a full body motion capture suit (see Fig 1.1). Markers are attached at pivot points or connections between rigid parts of the subject. A number of cameras then record the body's motion as it acts out the movements required for a scene or sequence. The recorded motion is then passed to a motion capture program which only sees the markers and excludes everything else. A skeleton of the body in motion can then be formed and animated with the resulting data. This can dramatically reduce the cost of animation.

1.2 Motivation

Motion is one of the most important aspects when watching movies or playing video games. The more lifelike the movement of the characters, the more the viewer is able to be immersed in the animated world. Obtaining realistic motion usually involves key framing, physically based modelling or motion capture.

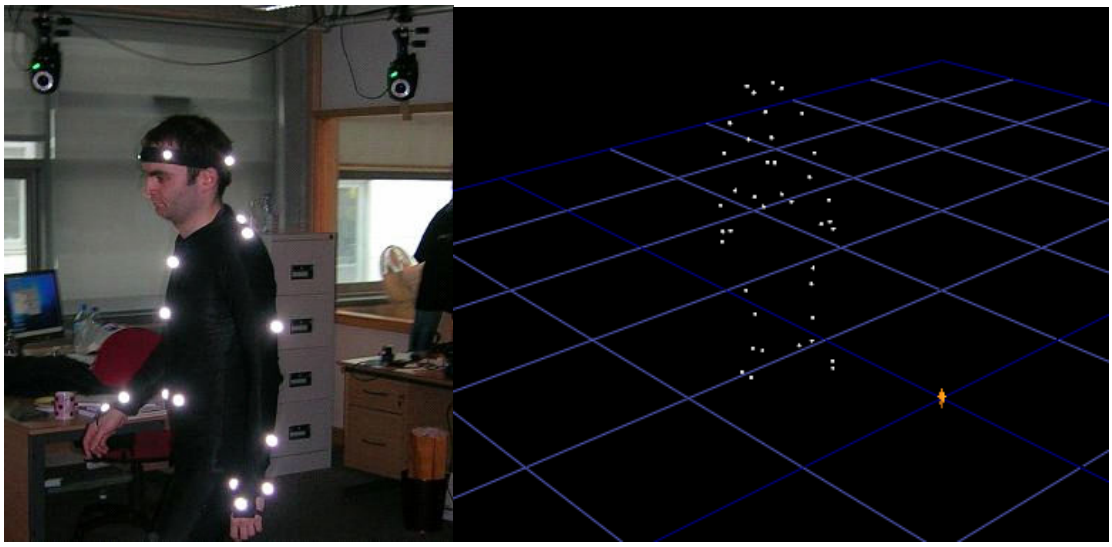


Fig 1.1 (left) Photo of the author in a motion capture suit with white markers attached. (right) Computer representation of the markers in three dimensional space.

1.2.1 Key framing

Key framing comes from the notion that a body has a beginning state and then a number of other key states. The idea is that the frames in-between these states can be figured out from these key frames. In hand-drawn animation, the senior artist would draw these frames, and leave the in-between frames to junior animators. In computer animation, the animator creates only the first and last frames of a simple sequence; the computer fills in the gap. Key framing requires a lot of effort and expertise. As well as that, for the motion to become close to realism, a large number of key frames are required.

1.2.2 Physically-Based Modelling

Physically-based modelling involves the use of natural properties of systems to determine the parameters of motion in an animation. For example, you could model the effects of collisions between billiard balls on a pool table using relatively simple mechanics. Unfortunately, realistic human motion is difficult to simulate using this technique.

1.2.3 Motion Capture

People are incredibly adept at noticing inaccuracies in human motion. This is why we use motion capture. Motion capture ensures that the characters movements are realistic as each frame of motion is taken directly from the motions in the real world. However, it has two main drawbacks:

1. It is very expensive and time consuming for actors to record each individual piece of motion. As well as this, the motion is rarely reused in subsequent sequences.

2. Even after a piece is recorded, it may not fit the requirements exactly, e.g. synchronizing motion to a background shot.

Motion capture is a realistic way of *reproducing* motion but not *creating* it. In order to make motion capture practical, it needs to be made reusable. We must use previously captured data to generate new original motions to animate our characters. Motion capture data has proven to be very difficult to modify. Editing motion capture data is a problem "analogous to that of editing a bitmapped image or sampled hand-drawn curve" [WP95] (see figure 1.2).

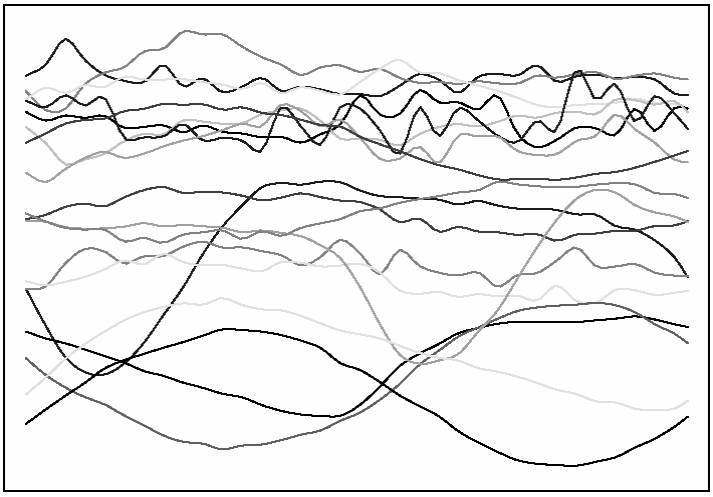


Figure 1.2: Captured motion curves of human walking. From [WP95]

1.3 Goals

This project aims to investigate 3D animation theory and techniques and to investigate motion capture techniques and methods needed to make it reusable. The project also aims to provide a software tool for synthesising motion, based on a collection of previously recorded motion capture data. The system is based on the paper "Motion Graphs" by Lucas Kovar, Michael Gleicher and Fred Pighin [KGP02]. Finally the project aims to evaluate the application and the results produced.

1.4 Readers Guide

The following will give a brief outline of this report, including a short summary of each chapter:

Chapter 1: Introduction of the report giving an outline of its motivation and goals, and a short summary of what's to come in Reader's Guide.

Chapter 2: Brief background and history of computer animation, performance animation and motion capture followed by details of Motion Graphs concepts and related papers.

Chapter 3: Details the design and implementation my program.

Chapter 4: Evaluation of each aspect of my program and any results produced.

Chapter 5: Overall project review including conclusions and future directions.

Appendix A: Details on some of the motion capture formats I looked at.

Appendix B: Details on some of the tools and frameworks used in the implementation.

Appendix C: Some code and data from the program as referenced throughout this paper.

Chapter 2

Background and Related Work

2.1 Introduction

Articulated figure animation has become popular in recent years because of the desire to use human beings as synthetic actors in three-dimensional computer animation environments [WW92]. Notable is the film *Rendezvous à Montréal* (see [TH87]) which features computer generated versions of Marilyn Monroe and Humphrey Bogart. The director Nadia Magnenat Thalmann comments "Actors will soon be out of a job…. film makers will be able to create characters so lifelike that members of the audience will not be able to distinguish them from real actors.", [TH87]. This comment made in 1987 has not been realised as of yet, however computer generated characters have become increasingly popular over the past ten years.

Early attempts of computer animated movies and characters included *The Works* [WI78] a feature length movie, of which clips were shown at SIGGRAPH in 1982, but was never completed. *Max Headroom*, a supposedly fully computer generated character appeared on British television screens in 1985 presenting music videos. The character went on to have his own talk show in the US. Max Headroom, however, was revealed to be a counterfeit, and was simply the actor Matt Frewer in a latex and foam rubber prosthetic makeup, superimposed over a moving geometric background. 3D rendering technology in the mid-1980s was not sufficiently advanced for a full-motion, human head to be practical. Max Headroom, however, proved to be incredibly

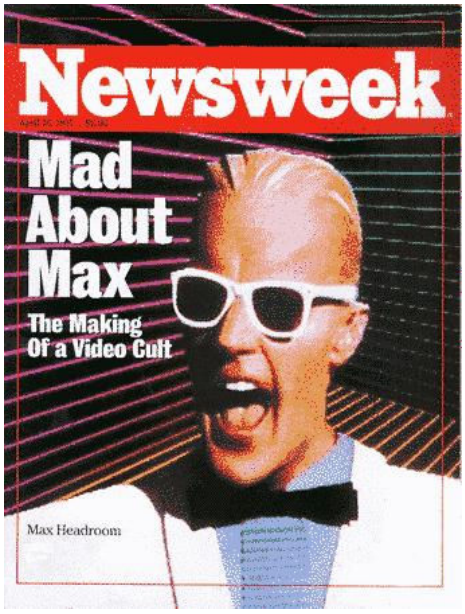


Fig 2.1 Max Headroom popularised the idea of computer generated characters, even though he was not computer generated.

successful and developed the public's interest in computer generated characters.

It was not until 1995 that *Toy Story* [LA95], the first completely computer-animated feature opened in cinemas. Pixar's film, which featured toys as the main characters, became one of the highest grossing movies of all time. Since Toy Story, a number of other computer generated movies have been released including, Monsters Inc., The Incredibles, and Cars, all produced by Pixar and Disney and Antz, Shrek, and Madagascar, all produced by Dreamworks.

All of the aforementioned movies use skeleton structures on which to base the movement. Most animation of "clothed" structures is controlled by animating an underlying skeleton, and then rendering the final images with flesh or clothes. Early work in this area was done by David Zeltzer [ZE82]. He designed a hierarchical control system that allowed the user to control the movement of a skeleton. He showed that high level control of the movements of the skeleton simply requires variables specifying the speed and duration of the movement.

2.2 Performance Animation

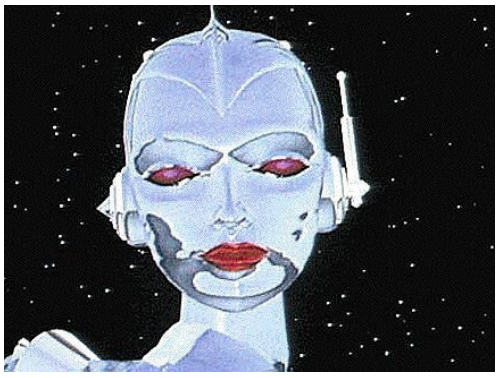


Fig 2.2 Shot of the robot from Robert Abel's commercial Brilliance (1985). This is considered to be the first use of motion capture

In the late 1800's, Marey and Muybridge independently conducted studies of human and animal motion by taking multiple photographs of subjects over a short period of time. These studies have had made an impact in a number of disciplines such as biology, medicine and of course animation [ME99].

Thirty years later the cartoonist Max Fleischer used this idea to develop the rotoscope. This was a device which allowed animators to trace characters motions from photos of real life performances. This was the first case of a real-life performance being used to create an animated character. This concept came to be known as performance animation.

The next step in performance animation was Robert Abel's commercial for canned food from 1985 entitled "Brilliance" (see Fig 2.2). This commercial marked the birth of 3 Dimensional motion capture for animation.

Each of these stages went through phases in which many people in the animation industry believed they would replace traditional animation with these methods. The rotoscope never did and to this day it is not widely accepted but is still used in instances to achieve a certain style.

Motion capture is still at a stage where it is being tested and developed as a production tool. It is nowhere near the stage of replacing animators and many believe that it never will. Either way it is still a very powerful tool for creating detailed animation, and will always have a use in producing a style of motion in characters.

2.3 Motion Graphs

The concept of Motion Graphs takes a step towards the goal of directable motion capture. The idea was proposed by Lucas Kovar, Michael Gleicher and

Fred Pighin in their paper from 2002 [KGP02]. The paper presents a method for synthesizing paths of motions based on a corpus of motion data. Given a set of motion capture data, a structure called a Motion Graph is compiled that encodes how the captured clips may be re-assembled in different ways. The motion graph is a directed graph where nodes correspond to clips of motion. The nodes then serve as segments of a continuous path to be followed. As the method automatically detects paths between motion data segments, users need not capture motion data specifically designed to connect to one another.

The Motion Graph idea turns the motion synthesis problem into one of selecting sequences of nodes, or *graph walks*.

2.4 Related Work

Recent years have seen a number of algorithms and ideas for motion synthesis from motion capture data. The idea of connecting clips of motion data was independently considered by Kovar et al in "Motion Graphs" [KGP02] and Arikan and Forsyth in "Interactive Motion Generation from Examples" [AF02]. Both of these had similar approaches, but both differed in their methods for matching frames and generating motion from the graph. [AF02] looked for solutions to problems which require "global planning", e.g. to be able to jump at a particular moment. It also used Hard and Soft constraints on the motion. These describe what *must* and what *should* occur respectively.

"Snap Together Motion" [GSKJ03] constructed motion graphs with a small number of transition nodes, but many links. The idea is to break motion capture data into a set of short clips. A user guided process then selects "common" character poses. The system automatically synthesises multi-way transitions that connect through these poses. The user is provided with an interface to construct a motion graph with "hub nodes" which have a high number of incoming and outgoing nodes. This paper uses the concept of constructing graphs with only a few transition nodes, but many links, which ensures that any trajectory can be generated from the graph.

"Motion Synthesis from Annotation" [AFO03] allowed the user to assemble motion into a timeline with annotations like *walk*, *run* or *jump*. The vocabulary for the annotations is chosen by the user. The system then assembles frames from a motion database so that the final motion performs the specified action at specified times. For example the user may describe a motion like "running while carrying but not jumping". The "not" specifies a negative annotation so that the algorithm is prohibited from generating undesired types of actions.

"Evaluating Motion Graphs for Character Navigation" [RP04] takes the focus of motion graphs away from the "local" quality of the motion (e.g. creation of

good blends and transitions between frames) and instead focuses on more global problems such as the ability to effectively follow paths. The goal of this paper is to create a character that can travel completely through the environment while maintaining natural looking motion. The paper suggests using evaluation metrics on the quality of paths travelled through the environment and ability to reach any position and orientation.

"Group Motion Graphs" [LCF05] is a technique for animating groups of skeletons, such as, flocks, herds or crowds. The idea is to have two forms of controls; one over what the group looks like, and one over what the group does. For example, an appearance goal could be to resemble a particular type of herd, while an action goal might be to follow a particular path through the environment. The idea of obstacle avoidance is also included in the paper. This paper uses a similar concept to the idea in [GSKJ03] of constructing graphs with only a few transitions, but many links, to ensure that any trajectory can be generated from the graph.

2.5 Research

A number of books and online resourses were consulted during my research of Motion Capture and Computer Animation. "Advanced Animation and Rendering Techniques" [WW92] details the animation of articulated structures. Basic concepts such as forward kinematics as well the history of the topic are discussed. Books such as "Computer Graphics using OpenGL" [H90] and "3D games: Real-time rendering and Software Technology" [WP01] were consulted for information on matrices and OpenGL concepts during the initial phases of the project. Other OpenGL advice came from John Dingliana's notes for his course 3D4 as taught in Trinity College in 2005. Jim Blinn's paper "Simulation of Wrinkled Surfaces" [B78] detailed concepts relating to local minima and partial derivatives. "Understanding Motion Capture for Computer Animation and Video Games" [ME99] offers an excellent introduction to the history and ideas involved in motion capture.

2.6 Motion Graph Construction

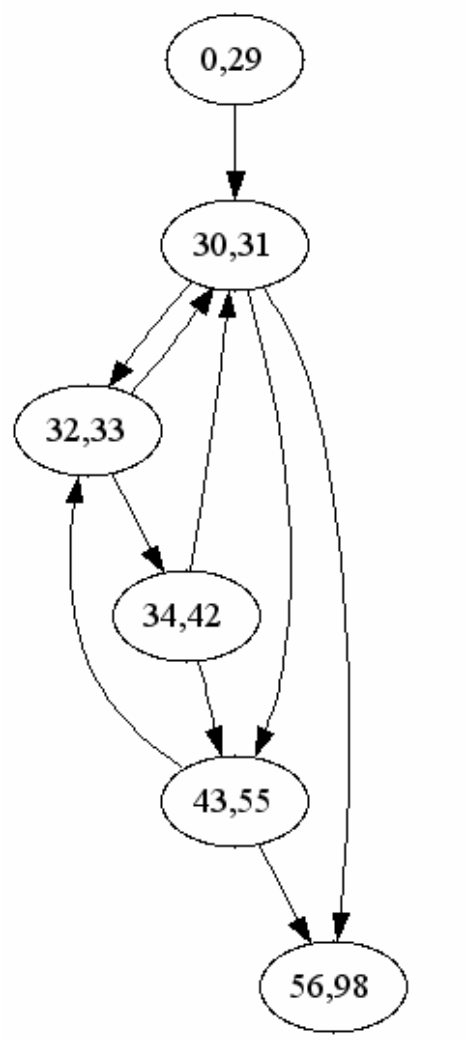


Fig 2.3 Motion Graph

A Motion Graph is a directed graph which connects clips of motion. A trivial motion graph can be created by placing all the clips from the database in their original order as arcs in the graph. For example in Fig. 2.3 we have a total of 99 frames of animation [0-98]. Starting at the 0^{th} frame we may play segment [0-29] followed by [30-31] and so on up to [56-98].

However more interestingly, after we have played segment [34-42] we can then hop back to segment [30-31] and loop over segments motions. For a segment to have multiple outgoing paths there must be multiple segments that can follow on from it. In order to find which hops can be made between segments we must be able to compare frames at the edges of clips in order to make smooth transitions. We use an error function to do this.

2.6.1 Error Function

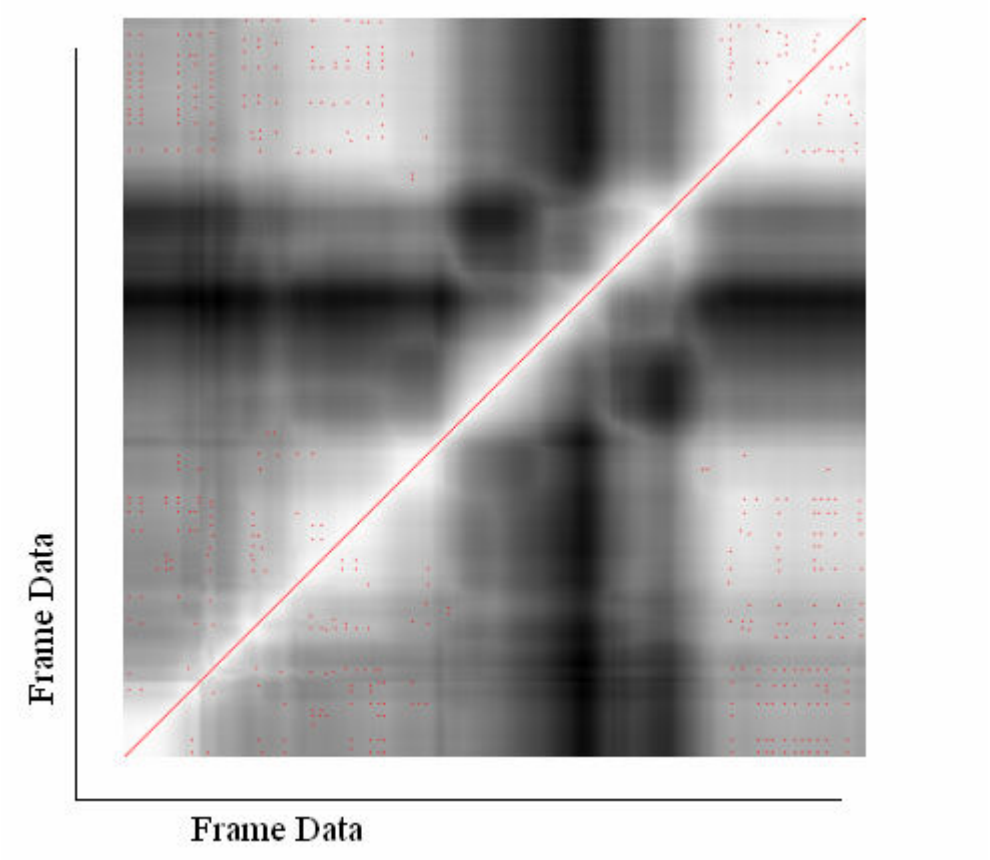


Fig. 2.4 An example error function. The entry at (i , j) contains the error for making a transition from the ith frame to the jth frame.

The error function compares frames within the motion data and produces a number which we use to determine if a transition from one segment to another can be made.

From this function we can create a graph of each frame of the data plotted against each other. We use this to determine if a smooth transition from the end of one segment to the start of another can be made. On the graph, white values correspond to lower errors and black values to higher errors. The coloured dots represent local minima. Along the diagonal are trivial local minima as a frame plotted against itself will have error zero.

Smooth blends require more information than can be obtained at individual frames. A seamless transition may need information about velocity or acceleration of joints which would be obtained from sequences of frames.

Details of the actual error functions investigated are given in Chapter 3.

A local minimum in the distance function does not necessarily imply a high quality transition; it only implies a better transition than the surrounding neighbours. We want local minima with small error values. The simplest

approach to this is to only accept local minima below a user determined threshold.

From this information we can assemble our motion graph. The motion capture data is broken into segments and paths are automatically generated between them. The animated character can then be put in motion using a sequence motion data as determined by the error graph.

Chapter 3

Design & Implementation

3.1 Overview

The purpose of this project is to provide a system for displaying animated motion capture data and to provide in this implementation the “Motion Graphs” idea based on [KGP02].

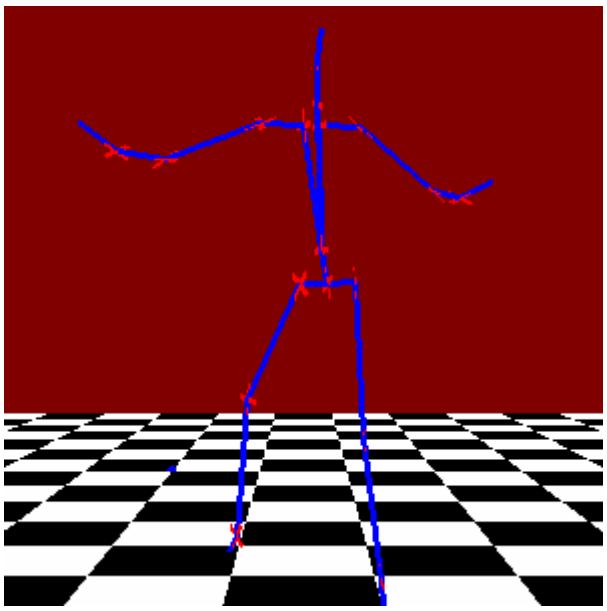


Fig 3.1 Initial pose of skeleton being animated as displayed by program.

3.2 Design Choices

In order to develop a program for displaying motion capture, I had to make some choices regarding the tools to be used for this. I had some background in using the OpenGL graphics libraries and I felt they would suit the simple 3D environment I was trying to create. There were also a number of different Motion Capture file formats to choose from (see Appendix A). I decided on Biovision's BVH format based on the number of files available on the web, as well as its simple easy-to-read ASCII style. I also looked at a number of Motion Capture file viewers including "Motion Capture Viewer" by David Zier [Zie02], "Dagger"by Jeff Lander [JL] and a motion capture program written by Michiel van de Panne of the University of British Columbia [MVP].

3.3 BVH parsing

In order to extract the data from the BVH file so that it could be used to animate a skeleton, the file must first be read by the program. In order to do this I modified a number of methods which were taken with permission from a motion capture program written by Michiel van de Panne [MVP].

First the skeleton's hierarchy is read in joint by joint from the BVH file. The C command "`sscanf`" is used to read the data from the file. As each joint is read in, the offset and channels for the joint are also stored in memory. The root joint is the first joint created and each child joint is stored as a member of its parent.

Once this is done, we allocate memory for the actual motion capture data based on the number of frames and joints. The data for each channel is then read and stored.

3.4 Display

Having looked at other motion capture viewers, I decided a simple stick man was the best approach to display the motion capture data being animated. This would keep the expense of rendering each frame to a minimum. As in Jeff Lander's program, I decided to mark the intersection of each joint with an X to allow the user to easily identify the start of each joint.

3.4.1 Bone Drawing

As the file is parsed, each joint of the skeleton defined in the BVH file is drawn recursively. Using OpenGL commands we specify the line width and colour to draw the bones. The bones length is defined by the "OFFSET" as given in the BVH file.

```
glLineWidth(3);
glColor3f(0,0,1);
glBegin(GL_LINES);
glVertex3f(0,0,0);
glVertex3f(offset[0],offset[1],offset[2]);
glEnd();
```

Once we have drawn the joint we translate to the joint's edge, so any child joints can be drawn.

```
glTranslatef(offset[0],offset[1],offset[2]);
```

3.4.2 Background

The background consists of simple red "wallpaper" and a black and white tiled floor. The floor is drawn using a simple algorithm. Each time we draw a tile we flip the colours. Incidentally, this tricolour look was chosen for aesthetic reasons to resemble the 'red room' in David Lynch's "Twin Peaks".

```
for (zcount=0; zcount<=nsteps; zcount++) {
      for (xcount=0; xcount<=nsteps; xcount++) {
            glColor3f(colour,colour,colour);
            glBegin(GL_QUADS);
            x = xmin + xcount*xsize;
            z = zmin + zcount*zsize;
            glVertex3f(x,0,z);
            glVertex3f(x+xsize,0,z);
            glVertex3f(x+xsize,0,z+zsize);
            glVertex3f(x,0,z+zsize);
            glEnd();

            //This just makes the colours flip on the tiles
            if (colour==1){colour=0;}else{colour=1;}
      }
}
```

3.5 Skeleton Animation

Once the initial pose of the skeleton has been drawn, the position of joints as well as the position of the actual skeleton in the 3D environment can be altered and redrawn using the motion capture data from the bottom section of the BVH file.

3.5.1 Frame of Motion

To draw a single frame of motion, each bone is drawn as detailed above in section 3.4.1. Once a given bone is drawn we then check how many channels of motion data the specified bone has. A single bone may have either 3 or 6 channels. 3 channels mean that only rotations have been specified (about the x, y and z axes) for each frame of motion while 6 channels require a translation to be performed. Typically the ROOT bone will be the only one to have a translation. The ROOT bone is generally the hips, and the translation performed on it will specify its position within the 3D space. Other bones will not need to be translated as the skeleton is a fully connected structure.

Once any translations have been performed, the bone is rotated into its position for the given frame. The rotations are performed using OpenGL's glRotatef command on the 3 channels of rotation data for each bone. Using a switch statement the bone is rotated depending on the axis the channel data in the channel is referring to.

```
case 'X':
      glRotatef(angle,1,0,0);
      break;
case 'Y':
      glRotatef(angle,0,1,0);
      break;
case 'Z':
      glRotatef(angle,0,0,1);
      break;
```

3.5.2 Playback

To animate the character we simply draw each frame of motion in a sequence.

```
for (int f=0; f<nframes; f++) {
...
      drawFrame(f);
...
}
```

As well as this a user may select to have a specific frame drawn, or when performing a graphwalk, to have the frames played back in another sequence. (See Section 3.10).

3.6 Camera Movement

The position of the camera (what the user sees) in relation to the skeleton must also be controllable by the user. To do this we manipulate OpenGL's `gluLookAt` matrix.

```
gluLookAt(eyex,    eyey,    eyez,
          centerx, centery, centerz,
          upx,     upy,     upz);
```

The first row specifies the position of the eye point (where the camera is located in the 3D environment), the second row specifies the position of the reference point (what the camera is looking at), and the third row specifies the direction of the up vector. I then use a matrix class and a vector data structures to perform operations on these values. These matrix and vector structures were supplied by John Dingliana as part of his 3D4 class.

To execute a simple motion of the camera, for example, say, a movement to the right, we declare `CamUP`, `CamForward` and `CamRight` vectors, and compute the values based on the current state of the `gluLookAt` matrix.

```
CamUP.set(upx, upy, upz);
CamForward.set(centerx - eyex, centery - eyey, centerz - eyez);
CamRight=cross(CamForward,CamUP);
```

The `CamRight` vector is the cross product of the `CamForward` and `CamUP` vectors. We can then move the camera left or right by adding or subtracting from this vector. This is done with the user controls. (See Section 3.12).

More complex camera controls like Pitch may also be executed; for example to pitch the camera upwards, we declare a temporary vector v, and initially subtract the eye point from the reference point (this moves the system back to its origin). We then set the vector v to be the reference point and rotate it around the CamRight vector by a given angle (0.5 in example). As we do this the camera can be made tilt upwards or downwards. After this process is complete we return the reference point back to its original position by adding on the eye point.

```
centerx = centerx - eyex;
centery = centery - eyey;
centerz = centerz - eyez;
v.set(centerx, centery, centerz);
v=rotation(0.5,CamRight)*v;
centerx = v.x();
centery = v.y();
centerz = v.z();
centerx = centerx + eyex;
centery = centery + eyey;
centerz = centerz + eyez;
```

Other camera options such as Yaw and Roll may be completed using similar methods.

3.7 Performing calculations

In order for frames to be compared using a graph of an error function, the actual function must be chosen. I constructed the functions myself, and modified based on how well each one performed. Combinations of error functions were also tested.

3.7.1 Angle Differences

The most basic of the error functions is the angle differences. Each frame is compared by adding up the absolute values of the differences between each of the angles of rotation. The rotations of the root joint are excluded in the comparison as they represent the rotation of the entire skeleton, and not positions of individual bones. We ensure we always take the acute angle by subtracting the value from 360° if it is greater than 180°. All of these values are added up and the result is an error value for two frames of data.

3.7.2 Angle Differences to a Power

Other error norms I tried included the Euclidean error norm (the square root of the sum of the squares of the differences).

$$d = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}.$$

In this formula x_i and y_i are equivalent angles in different frames. I also tried higher powers like the 4^{th} root of the sum of the angle difference to the 4^{th} power.

3.7.3 Angle velocities

Rate of change between positions of angles was accomplished by the following method. If we wished to compare an angle in frame x and frame y then we first

of all compared frame x to frame x-1 followed by frame y and frame y-1. We then compared the results of these calculations

3.7.4 Angle Accelerations

When comparing the rate of change of the accelerations we use similar calculations to those in the velocity calculations. In this case the differences calculated between frames x and x-1 and those calculated between x-1 and x-2 are subtracted from each other and compared to the equivalent value for frame y.

3.7.5 Attempted Error Functions

Details of these error functions are given in Chapter 5.

3.8 Displaying the Error Function

The graph of the error function is displayed by first creating a character array.

```
image = new unsigned char[nframes*nframes *4];
```

This array will store the image pixel data for the graph. The size of the array is the number of frames squared (as we are graphing each frame against each other one) multiplied by 4. The 4 represents the number of channels in the colour space of the image.

3.8.1 Calculating Local Minima.

Before each of the pixels in the image of the graph may be coloured, the local minima of the error values must be calculated. This is done using two methods.

3.8.1.1 Nearest Neighbour

For this method, I treat the error values as if they are placed at each point on pixel grid, of which the size is the square of the total number of frames. I then compare each error value to the surrounding error values. If this value is lower than each of the points that surround it, then the point is deemed to be a local minimum.

3.8.1.2 Partial Derivatives

For this method, we approximate the partial derivative along either the x or y direction of the point we are testing. To this we subtract the error value from each of its neighbours directly adjacent in the given direction. We take these values to be approximations to the slopes.

$$\frac{\partial f(x)}{\partial x} = \frac{f(x) - f(x-1)}{\Delta x}$$

If this value changes from positive to negative as it crossed the point, we then have a local minima.

3.8.2 Image Data

The actual image data must take values between 0 and 255. We do this using the following equations

```
errorlength = MaxError - MinError;
imagevalue = round(Value – MinError *(255.00/errorlength));
```

We calculate the distance between the largest error and the smallest error, as this interval will be represented by the numbers 0 up to 255. We then normalise the error values by first subtracting the smallest error value calculated, and then multiplying this by 255 divided by the errorlength. The result will be a value between 0 and 255 which is then passed into the image character array. The result of this is that high errors are darker than low errors.

Any values which are deemed to be local minima are coloured in red on the graph. A point is accepted as a local minimum if it is below the user defined threshold. Using the data in the character array we can then display the graph.

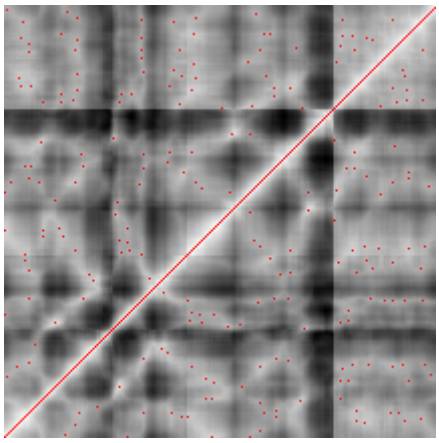


Fig 3.2 Graph of Error Function

The user is also given the option to display the data using Gnuplot. In this case the program makes a system call to Gnuplot and passes it the data to be displayed.

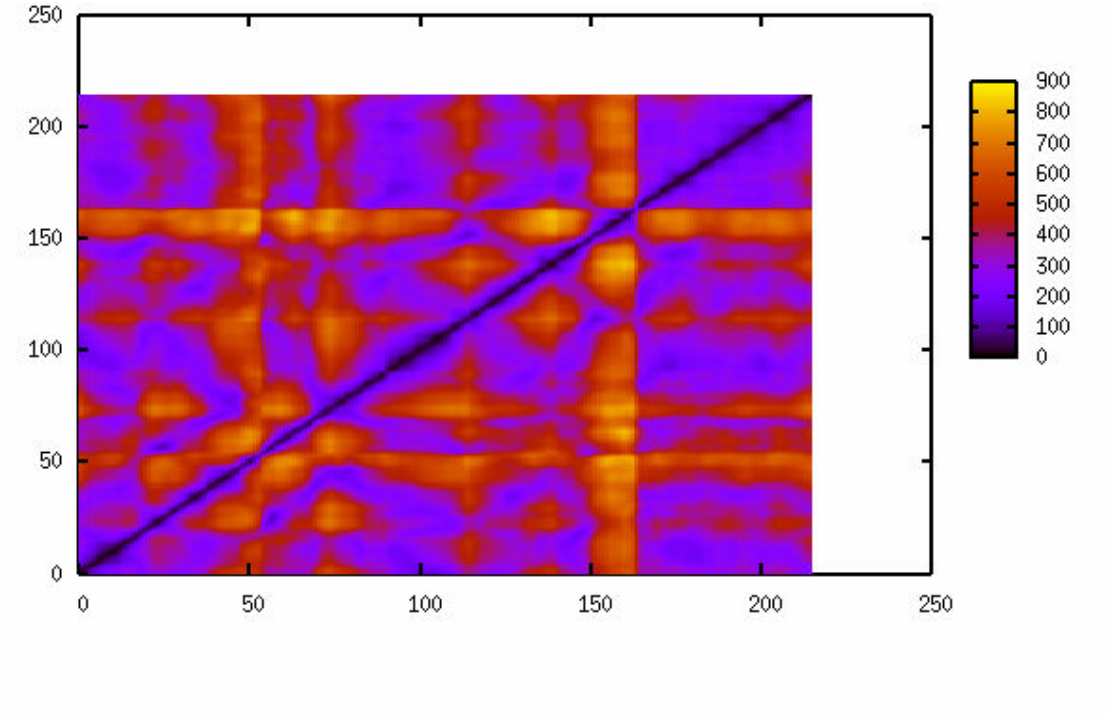


Fig 3.3. Gnuplot graph of error function

3.8.3 Graph Options and Graph Looping

By clicking on a point on the graph, the user may create loops for the skeleton to walk through. For example, if the user sees that the point (i, j) is a local minimum on the graph, he may click on the point which will result in the following action:

Left Button
This will animate the skeleton by looping over frames 0 up to i followed by frames j up the last frame.

Right Button
This will animate the skeleton by looping over frames i up to j continuously. This option is particularly useful for creating loops of repetitive motions like walking.

As well as these options, the user may select to have a motion graph created, or to have a graph walk performed. (See Section 3.12)

3.9 Creating the motion graph

The motion graph is a directed graph so to display possible paths through the motion data, an open source network visualisation program called Graphviz was used (see Appendix B). The Dot system in Graphviz takes descriptions of graphs in a simple text language. An example of a motion graph written in this language and the resultant motion graph is given below.

```
digraph motiongraph
{
"0,20"->"20,50";
"50,70"->"70,100";
"20,50"->"0,20";
"20,50"->"50,70";
"50,70"->"0,20";
"0,20"->"70,100";
}
```

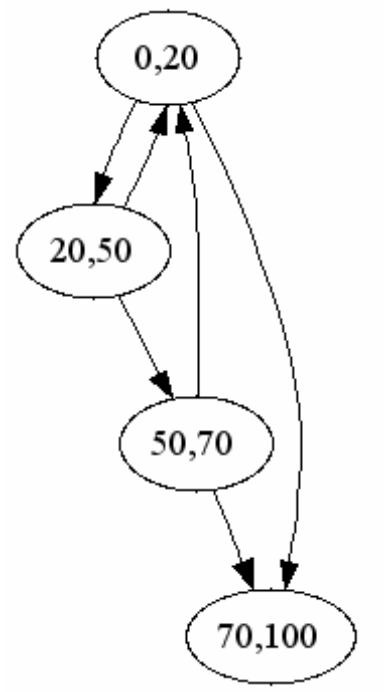


Fig 3.4 Motion Graph

The image is output as a PNG image and can be displayed by most image viewers.

3.9.1 Path finding

I will describe my method for creating the directed graph by example. Given a set of data which is 99 frames long (0-98), say we set a low threshold on the error function and we are given three local minima. The local minima will be stored in an array called localmin, as pairs of integers in order of the first number. If 12,23 is stored then 23,12 will also be stored.

0	12,23
1	18,45
2	18,70
3	23,12
4	45,18
5	70,18

In this example the directed graph is created by first creating the segments of motion capture data. In this case the segments would be frames 0-12,12-18,18-23, 23-45, 45-70 and frames 70-98. Finally using the localmin data, we create paths through these segments. This is then output to a file in the standard Graphviz language.

The commented code for the algorithms used is given in Appendix C. The comments explain the problems which can arise during the creation of the directed graphs.

3.10 Performing a Graph Walk

Graph Walks are performed by cycling through the data in the localmin array and finding points to jump to. Certain restrictions can be put on the choices of jumps to accept. By default each time it plays a segment, at least five frames are played before a suitable jump is chosen. When a graph walk is performed a text file is output contain details of the path taken. The commented code for the algorithms used in this is given in Appendix C

3.11 Separation of the Root Node

The rotations and translations of the root node must be separated from the motions of the skeleton if frames of motion are to be played back out of sequence. At different points in time in the motion the skeleton may be facing different directions or may be at a different point in space. As a result of this, the user is given the option of fixing the rotations and translations of the root node to the values they have in the first frame.

3.11.1 Walking in Original Path

The skeleton can be made follow its original root translations (from the 0^{th} frame up to the final frame), while doing a graph walk. This is done by having the root bones motion data taken from sequential frames while taking the other bones motion data from other sequences of frames from the motion graph. This allows a graph walk to be performed while still retaining the original curve of the path taken by the skeleton.

3.11.2 Walking in a Straight Line

The user is given the ability to have the skeleton walk in a straight line, even if the original motion was curved. To do this we calculate the distance moved in each frame of motion in the 2 dimensional plane of the x and z axes (as y is the up direction).

$$d = \sqrt{(x_2 - x_1)^2 + (z_2 - z_1)^2}$$

x_2 and z_2 are the values from the current frame while x_1 and z_1 are the values from the previous frame. The skeleton can then be moved this distance in a direction chosen by the user.

3.12 Controls

The controls, to allow manipulation of the skeleton, were implemented using GLOW (see Appendix B). Having never written a program which needed user controls like those required, a number of frameworks were looked at including the GLUI and the GLFW frameworks. However the GLOW framework was chosen due to the simplicity of the controls required, as well as the fact that GLOW was designed with OpenGL programs in mind.

3.12.1 Main Controls

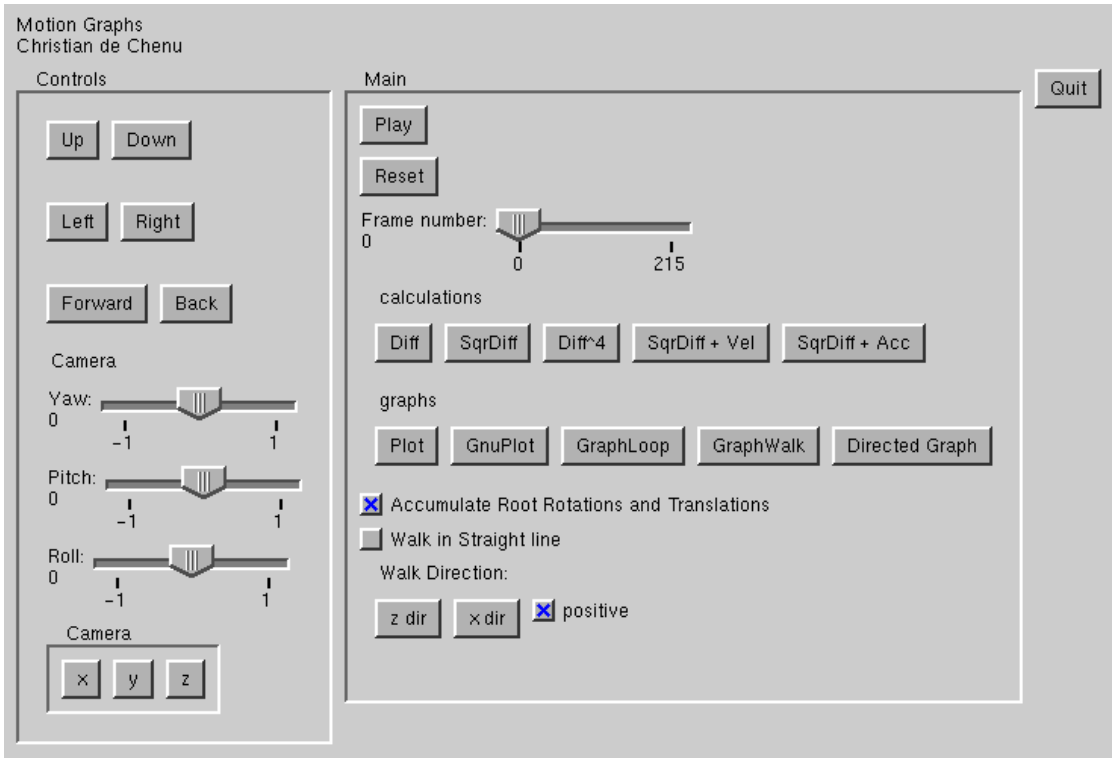


Fig 3.5 User Controls

The Left panel contains all the controls for manipulation of the camera position. At the top are the basic camera controls followed by sliders

controlling Yaw, Pitch and Roll. Below are buttons to reset the camera along the x, y and z axes.

The Right panel contains the Main Controls which are described below.

Play
Play back each frame of the motion capture data by animating the character.

Reset
Resets the character back to its initial pose and position.

Frame Slider
Allows the user to shuttle through each frame of motion.

Calculations
Allows user to perform calculations for generation of the error graph (Section 3.7).

Plot
Plots the error function data as a 2 dimensional graph, as well as opening up control options for the graph (Section 3.8).

GnuPlot
Plots the error function data using the program GnuPlot (Section 3.8).

GraphLoop
Plays back the looped motion as selected by the user on the motion graph (Section 3.8.3).

GraphWalk
Plays back a walk through the graph (Section 3.10).

Directed Graph
Opens the PNG image of Motion Graph using a program for displaying images (by default it uses the Windows Picture Viewer) (Section 3.9).

Accumulate Root Rotations and Translations
Toggles between whether the root node's translations and rotations should accumulate over the course of the motion (i.e. should the actual skeleton walk on the spot or move around the 3D environment.) (Section 3.11).

Walk in Straight Line and *Walk Direction*
Allow the user to determine the path taken by the skeleton in the 3D environment (Section 3.11).

3.12.2 Error Graph Controls

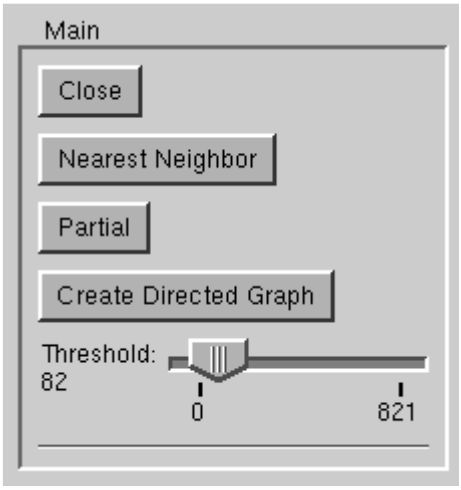


Fig 3.6 Controls for the Error Graph.

These controls are opened when the *Plot* button is pressed on the main controls.

Nearest Neighbour
Selects to use the Nearest Neighbout method of finding Local Minima (Section 3.8.1.1).

Partial
Selects to use the Partial Derivative method of finding Local Minima (Section 3.8.1.2).

Create Directed Graph
Creates a file in the GraphViz language that can be used to plot a directed Motion Graph.

Chapter 4

Evaluation & Results

4.1 Overview

The intention of the motion graph idea is create realistic controllable motion. When talking about motion being realistic, it is the human eye which is the best judge. This is especially true when it comes to *human* motion. As a result of this, the procedure for testing my program was a mixture of trial and error in the choice of function used, and personal reaction on the part of myself and other observers.

4.2 Motion Capture Data

One of the main problems encountered with this project was the lack of motion capture data. All of the data used in the project came from motion capture websites. The problem with this data that it was either too simple, (i.e. clips of a person walking, but with no variation in the walking) resulting in good results which are arguably trivial, or else the clips were too complicated (i.e. clips of a person constantly changing speed and direction) which would not be suitable for initial testing of the program. In order to get suitable motion capture data, a motion capture session was recorded in the Interaction, Simulation and Graphics Lab in Trinity College (see Fig 1.1). The session was recorded on a Vicon motion capture system. The data was encoded in the C3D format which is a point based motion capture format. The format I am using is the Biovision

BVH format which is a hierarchical skeleton format. While the conversion from one format to the other is possible, it involves defining the skeleton using software which was not readily available to me. However, the motion capture session proved to be a rewarding experience and was an important part in the study of motion capture.

4.3 Control of the Root Node

The separation of the rotations and translations of the root node was one of the first steps taken towards controllable motion capture. The method for this is explained in section 3.11. Initially all motions of the root node were set to their position in the first frame which resulted in the skeleton walking on the spot. The skeleton's motion was then looped over a sequence of frames (using the *Graph Loop* option). This option makes the root node use the motion data from sequential frames starting at the 0^{th} frame, while the other bones use data from the looped sequence of frames. For simple walking motions this proved to be

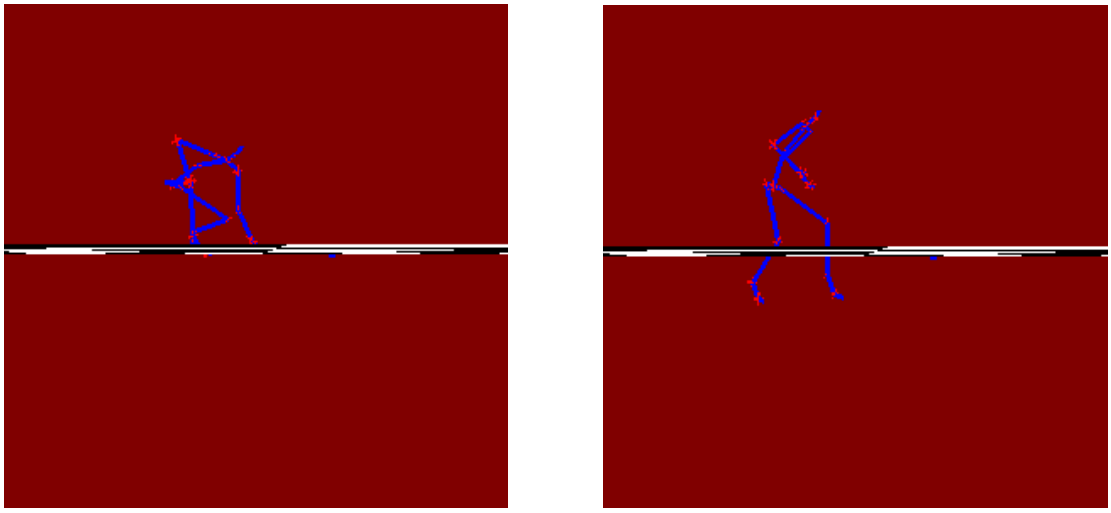


Fig 4.1 (Left) Frame 814 of motion data. All bones using the motion data for that frame. (Right) In this frame the root is using the motion data for frame 814 while the other bones are taking their data from another frame.

very successful. However for motions in which the skeleton changed speed or crouched down problems arose. If the skeleton's motion slowed down, while the motion of the feet was looping at a continuous speed, "foot sliding" occurred, and the skeleton looked like it was "moonwalking". If the skeleton crouched in the original motion, this meant that the root node was lowered which did not correspond to the continuous looped motion of the joints (see Fig 4.1).

4.3.1 Walking in a Straight Line

Having the skeleton follow the original path taken can be quite restrictive on the motion. As a result of this I included the option of having the skeleton walk in a straight line path, as chosen by the user. This method makes the skeleton

travel the distance that it would have moved in its original curved path. This prevented "foot sliding", as if the skeleton's motion in the original frame changed speed, its new straight line motion would also change speed (see Fig 4.2).

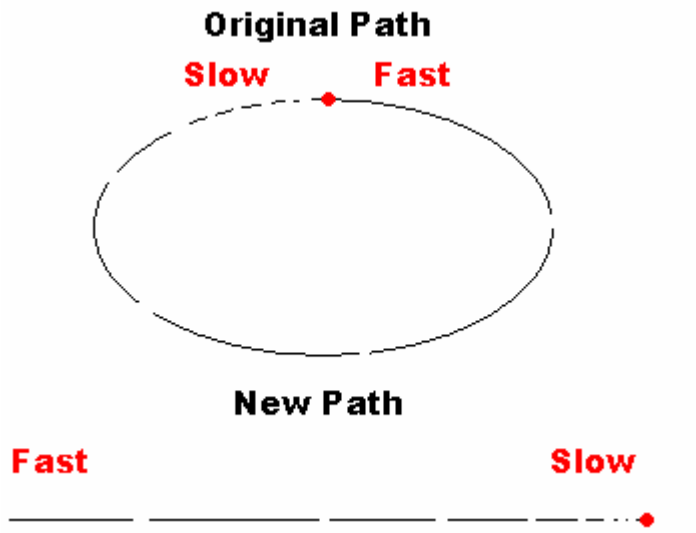


Fig 4.2 (top) Original curved path of motion with variable speed and (below) the new path which also changes speed.

4.4 Error Functions

For continuous motion to occur, an error function had to be chosen to give a smooth transition between frames. The first function to be implemented was the differences between angles. This only took the positions of joints into account and did not include any concept of velocities or accelerations. Simple motion data of walking and running proved very successful with this error function.

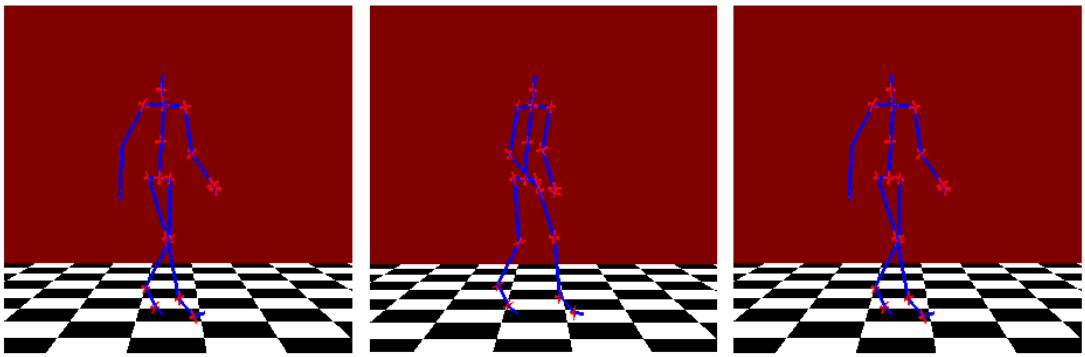


Fig 4.3 A simple walking motion where the skeleton returns to the same pose.

As the skeleton walked or ran, after a short sequence of frames, it would arrive at a pose close to that of one it held previously. As a result of this the function would give a low error result at these points.

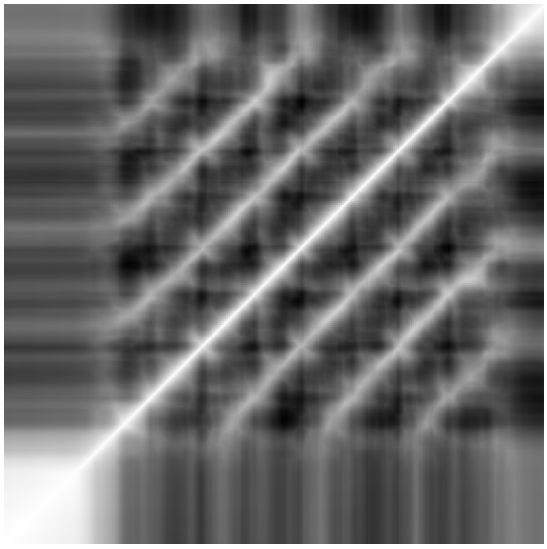


Fig 4.4 Sample graph of differences between angles. The white area in the bottom left hand corner is because the skeleton is stationary in the first few frame of motion.

The graph of this error function results in a repetitive pattern corresponding to the repetitive walking motion (see Fig 4.4). Following on from this function is the use of other higher order error norms. A number of these were tried; however the differences between the norms become less noticeable above the 4^{th} power.

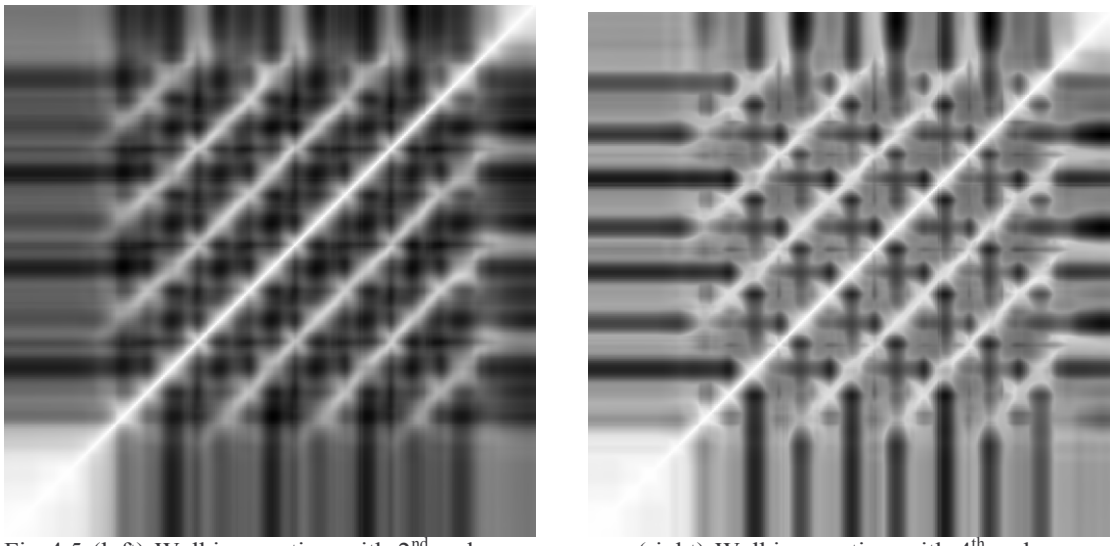


Fig 4.5 (left) Walking motion with 2^{nd} order error norm. (right) Walking motion with 4^{th} order error norm.

Results from these varied depending on what level the local minima threshold was set at. Since the setting the threshold on each function at the same level resulted in very different results, it was mainly trial and error which proved the most successful. As the order of the norm increased (ie to a higher power), the results became exaggerated as dark areas separated from white areas (see Fig 4.5).

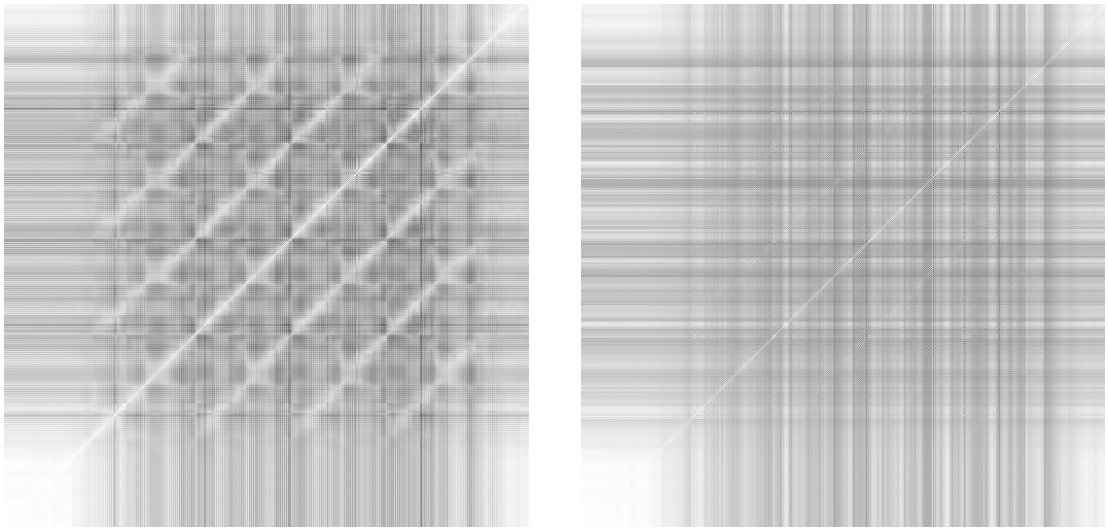


Fig 4.6 (left) Sample graph of velocity differences. (right) Sample graph of acceleration differences. In each of these images, outliers have been removed.

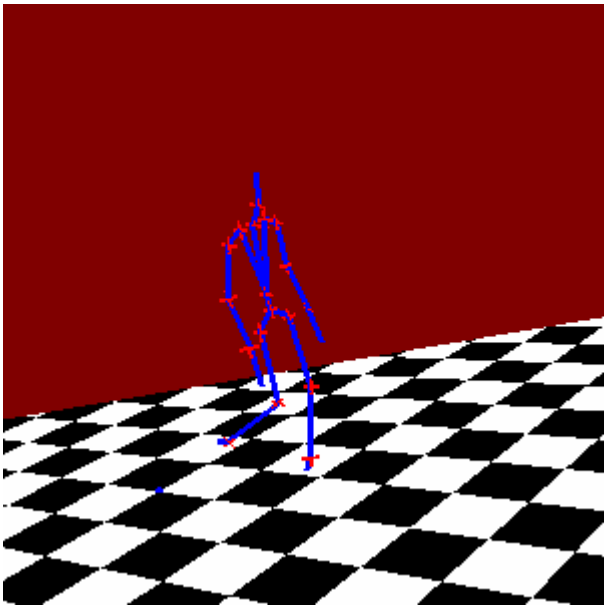
Results from the functions which used velocities and accelerations, proved less successful on most of the data I had. This is mainly due to the fact that many of the motions I used were walking or running which meant that acceleration was more or less constant between each step apart from changing suddenly at the end of a step. As a result of this I have removed outliers from the data when viewing the graph, as most of the graph appeared white.

4.5 Graph Walk

The system for automatically finding a path through the motion graph proved very successful. The minimum setting of five frames being played before a suitable jump found, produced some good results. Complex sequences of frames can be found with little difficulty. The fact that the paths were automatically generated has some negative aspects as well however, as the user has very little control over motions produced. For example, a smooth graph walk of a skeleton, constantly slowing down and starting up, was produced using the angle difference error function. Even though the motion was smooth, the fact that the skeleton constantly changed speed did not correspond to realistic human actions. Other error functions could ease this velocity problem, but better user control over the type of motions the skeleton executes, would make the graph walk much more practical.

4.5.1 Multiple transitions

Use of a high threshold, results in a large, complex motion graph (see Appendix C for an example of the motion graph produced). As a result from this, some successful results can be found with complex paths though the motion graph (see Fig. 4.7).



250-264
357-368
274-313
221-229
323-356
265-273
369-374
189-199
384-394
210-220
314-322
230-237
421-437

Fig 4.7 Short simple sequence like this man walking backwards can be produced involving a large number of transitions. The frame sequences are shown on the right (See Appendix B for the motion graph produced).

4.6 Creating new motion.

One of the key ideas of this project is the creation of new motion from old motion. Most of the successful results produced had to do with extending sequences of motion, rather than transforming the motion into something else. One successful attempt at a transformation was a clip of a dog hopping up and down. By locking the root node in place and trying a number of different thresholds on the error functions, it was possible to create a sequence of motion which resembled a dog running (See Fig 4.8).

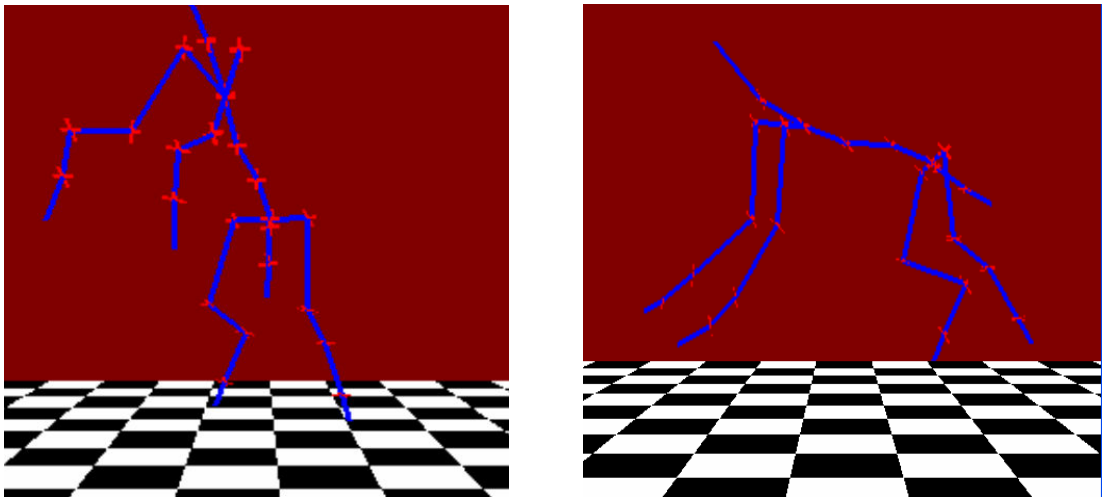


Fig. 4.8 (left) Original clip of dog jumping up and down and (right) the new sequence with the dog running. In this clip the root node of the dog is set at the motion data from the first frame.

The problem with this was the lack of translational data available for the root node. The original translation data mainly involved movement up and down in the y direction. The new motion required translation to be performed in the x and z directions.

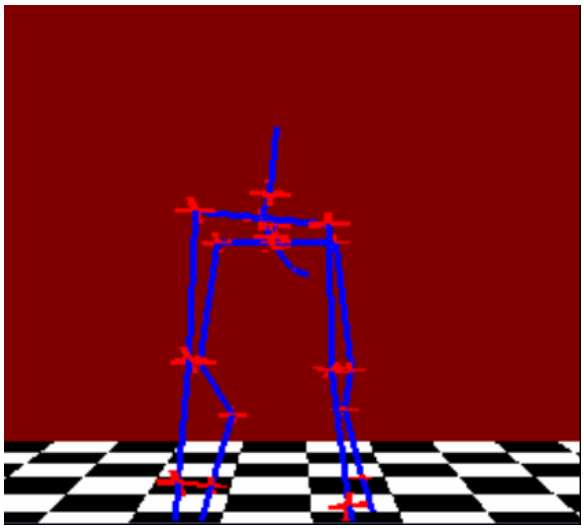


Fig 4.9 Clip of dog jumping at camera

Another clip was produced which when combined with the original translational data of the root node, looked as if the dog was jumping at the camera. While this result was reasonably successful, the dog appeared at points as if it was floating. The translational data was disconnected from the actual motion of the dog. Although this may work in certain instances; it is easy for the human eye to tell there is something strange about the motion.

Chapter 5

Project Review

5.1 Overview

The idea of controllable motion capture is still in its infancy and neither the movie nor the video game industry has started using it yet. The motion graphs idea, and this project as a result, are meant only to take steps towards the final goal. Though I feel all of the goals of this project were reached, there is still potential for improvements over the techniques that I used, as well as a lot scope for new ideas related to the motion graphs concept.

5.2 Achievements

The initial goals of the project were research based, and proved to be quite entertaining, as well as informative. Personally, I have a keen interest in movies. Motion capture is a concept that I was very familiar with from watching documentaries on the making of movies, and it was something that I enjoyed learning about.

The result of a user controlled system for playing motion capture data was completed successfully. As well as that, the inclusion of the error functions and the motion graphs was implanted suitably into the workflow of the program.

5.3 Problems Encountered

Over the course of this project, several problems were encountered. When it came to implement the program for displaying the motion capture data, a disproportionate amount of time was spent working on the control system. I am a mathematics student, and have never taken any classes in writing programs of a commercial quality. As a result I spent a lot of time working on the tutorials provided by the GLOW control framework.

5.3.1 Translational Data

Lack of translational data for the root node was problem, except in cases where the motion was extremely simple and in one direction. A solution to this would be to import longer sequences of translational data for the root node. However, even where the original sequence of frames was of a long duration, the resulting graph walk looked disconnected from the motions of the root node. Careful planning would be required to ensure that the actions performed by the individual bones of the skeleton corresponded to the motions of the root.

5.3.2 Motion Capture Data

Towards the final stages of the project it became apparent that more motion capture data would be required. Although a motion capture session was recorded, this data proved unsuitable (see Section 4.2). However most of the successes and failings of the project can be shown using the basic data that was available to me from the internet.

5.4 Conclusions and Future Directions

The best results from this project came from repetitive motions like walking and running. These results would be suitable for crowd scenes in movies and video games where the motion of a large number of background characters could be manipulated using motion graphs. This is especially true in video games where background characters may be moving perpetually.

Complex motions tended to look unrealistic. As well as that, when a number of frames from a complex series of motions was combined, the resulting motions often made no sense, or were something that a person would not realistically do.

The system of automatically generating paths through the motion graph should be supplemented by a user controlled system. This was an issue tackled in [GSKJ03] where the focus was on a user-interface that allowed the construction of a motion graph.

The original ‘Motion Graphs” paper [KGP02] blended frames between transitions and, although ‘jerkiness’ was not a big problem, this would help increase the fluidity of the motion.

The error function also has a lot of scope for improvement. Some of the ideas considered included weighting the error value depending on how many children were dependent on it, (e.g. part of the arm would get more weight than a finger) and weighting based on the length of the bone was investigated, but not implemented due to time constraints.

Appendix A

A number of file formats exist for motion capture data. However the Biovision (.BVA/.BVH) formats were the main ones I looked at.

Biovision's .BVA format

This is the simplest of the formats. This format is unusual in that even though a skeleton is defined, there is no hierarchy definition. Each bone is defined in its actual position. For each bone (or segment) in the skeleton, the segment name, number of frames and frame time are listed followed by nine channels of animation, representing the X, Y and Z values for translation, rotation, and scaling.

```
Segment:   Hips
Frames:    29
Frame Time: 0.033333
XTRAN      YTRAN       ZTRAN      XROT        YROT        ZROT       XSCALE  YSCALE  ZSCALE
INCHES     INCHES      INCHES     DEGREES     DEGREES     DEGREES    INCHES  INCHES  INCHES
0.000000   34.519684   0.000000   -14.988039  -12.240604  -3.481155  ...
0.102748   34.078739   3.159979   -15.337654  -14.320413  -3.983407  ...
0.260680   33.836613   6.487895   -16.308723  -15.090799  -3.861260  ...
...        REPEATS FOR A TOTAL OF 29 FRAMES
Segment:   Chest
Frames:    29
Frame Time: 0.033333
XTRAN      YTRAN       ZTRAN      XROT        YROT        ZROT       XSCALE  YSCALE  ZSCALE
INCHES     INCHES      INCHES     DEGREES     DEGREES     DEGREES    INCHES  INCHES  INCHES
0.272156   38.993561   -1.199981  -4.022753   -0.411088   1.354611   ...
0.413597   38.542671   1.932666   -4.371263   -0.591130   1.100887   ...
0.560568   38.279800   5.184929   -5.020082   -0.657020   0.768863   ...
...FOR THE REST OF THE SEGMENTS
```

Fig A.1 Sample Biovision BVA file.

Biovision's .BVH format

The .BVH format differs from the .BVA in a number of respects. This format is broken up into two parts, header section describing the hierarchy and initial pose of the skeleton and a second section containing the actual motion capture data. Each file starts with the word "HIERARCHY" followed by the word "ROOT" on the line bellow. After this is the name of the root (Hips in the example). A curly brace follows and within each braced block is the "OFFSET" and "CHANNEL" definitions. The offset defines the displacement from the bones parent.

```
HIERARCHY
ROOT Hips
{
      OFFSET  0.00     0.00    0.00
      CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
      JOINT LeftHip
      {
            OFFSET    3.430000           0.000000    0.000000
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT LeftKnee
            {
                  OFFSET   0.000000   -18.469999   0.000000
                  CHANNELS 3 Zrotation Xrotation Yrotation
                  JOINT LeftAnkle
                  {
                        OFFSET       0.000000   -17.950001   0.000000
                        CHANNELS 3 Zrotation Xrotation Yrotation
                        End Site
                        {
                                    OFFSET       0.000000  -3.119999   0.000000
                        }
                  }
            }
      }
...
}
MOTION
Frames:     20
Frame Time: 0.033333
 0.00  39.68   0.00     0.65   ...
...
```

Fig A.2 Sample Biovision BVH file.

Appendix B

OpenGL and GLUT

OpenGL (Open Graphics Library) is a cross platform API for writing applications that use 3D graphics. The interface consists of a number of different function calls which can be used to draw three dimensional shapes from simple primitives. GLUT (OpenGL Utility Toolkit) is a library of utilities for OpenGL programs. These were the core medium for displaying an animation of the character.

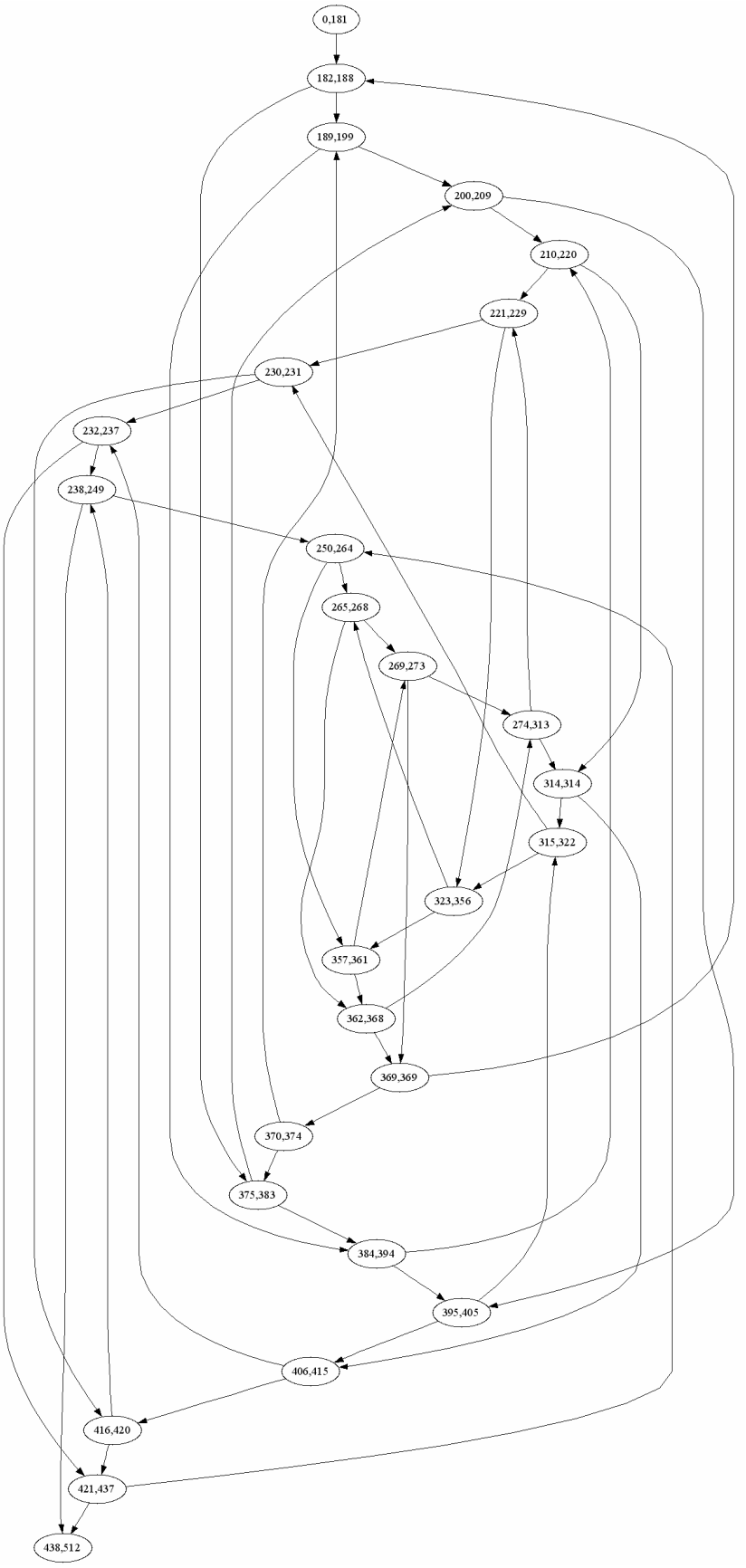
GLOW

The GLOW Toolkit is a cross-platform object-oriented framework for building interactive applications using OpenGL or similar APIs. It is basically a C++ wrapper for GLUT, providing an object-oriented API for creating windows, menus and other GUI elements, and for event handling. It also features a widget library for building user interfaces and full integration with OpenGL as the imaging API. The primary means of learning the interface was tutorials showing how to integrate GLOW into OpenGL programs, how to manage menus and understand GLOW's event reporting mechanism, and how to create a user interface using the buttons and widgets that GLOW provides. This had to be mastered before much of the work could begin.

Graphviz and Dot

Graphviz is open source graph visualization software produced by AT&T Research. I used the Dot system which makes hierarchical drawings of directed graphs. The layout algorithm aims edges in the same direction (top to bottom, or left to right) and then attempts to avoid edge crossings and reduce edge length.

Appendix C



Motion Graph produced in example from section 4.5.1

Code for creation of the Motion Graph

```
FILE *f = fopen("test.dot","w");/*This opens the file for writing ("w")*/
fprintf(f,"digraph motiongraph\n{\n");


//First we wish to include the segment zeroth frame up to the first jump segment.

// this is the point we wish to jump to
int tempend=0;

//If the point we are jumping to is listed twice (like 12,23 and 12,50) then we
move onto the last instance of it
while(localmins[tempend].i==localmins[tempend+1].i)
{
        tempend++;
}
fprintf(f,"\"%d,%d\"-
>\"%d,%d\";\n",0,localmins[0].i,localmins[tempend].i,localmins[tempend+1].i-1);

for(int i=0;i<localmincounter-2;i++)
{
//the following ifstatement rules out segments of 0 distance like (12,12) etc
        if(localmins[i].i!=localmins[i+1].i)
        {
                int tempendi=i;
                while(localmins[tempendi+1].i==localmins[tempendi+2].i)
                {
                        tempendi++;
//the cycle the end of the segment up the chain to avoid jumps like 10,12->12,12
                }

                fprintf(f,"\"%d,%d\"-
>\"%d,%d\";\n",localmins[i].i,localmins[i+1].i,localmins[tempendi+1].i,localmins[te
mpendi+2].i);

        }
}
//Finally we add in the last segment
fprintf(f,"\"%d,%d\"->\"%d,%d\";\n",localmins[localmincounter-
2].i,localmins[localmincounter-1].i,localmins[localmincounter-1].i,_bvhpointer-
>errorSize()-1);

//The basic sequence of segments have now been output to a file.
//Now we put in the jumps as defined by the local minima
for(int i=0;i<localmincounter-1;i++)
{
        //locate find the end of the segment we are jumping to
        int locate=localmincounter-1;
        //we set it equal to localmincounter-1 as we will search
        //backwards to get the last mention of the number
        while(((localmins[i+1].j)!=(localmins[locate].i))&&locate>-1)
        {
                locate--;
        }
        int start = i;

        while(localmins[start].i==localmins[start+1].i)
        {//so in this case there are >1 jumps from this frame, so we must find the
        //correct segment to jump from
                start++;
        }

        //start should now be the begining of the first segment
        //we should now have located it. we want to
        //jump to locate and locate+1 segement
        //note that i+1.j and locate.i are equal

fprintf(f,"\"%d,%d\"-
>\"%d,%d\";\n",localmins[start].i,localmins[start+1].i,localmins[i+1].j,localmins[l
ocate+1].i);
}
fprintf(f,"\n}");
fclose(f);/*And here we close the file*/
}
```

Code for performing graphwalk.

```
FILE *f = fopen("graphwalk.txt","w");/*This opens the file for writing ("w")*/

int dataframe=0;
int startj=0; //The start point of the segment. The j refers to the fact that we
              take the jth coordinate to begin the segment
int endi; //The end point of the segment. The i refers to the fact that we take the
            ith coordinate to end the segment
int stoppoint=0;

while(stoppoint<=nframes)
{
        int endi=0; //initially we set the end to 0.
        while(localmins[endi].i<=((localmins[startj].j)+5)&&endi<localmincounter)
        {
                //so we cycle through endi looking for a point to play up to.
                //the +5 makes sure we play 5 frames at least.
                endi++;
        }
        //Then we play the segment of at least 5 frames.
        //Once the frames are played we make the next hop from the i coordinate of
        //endi to the j coordinate. This then becomes startj and the cycle starts
        //again
        for (int j=localmins[startj].j;j<=localmins[endi].i;j++)
        {
                usespecialroottranslation = true;

                glClearColor(0.5f, 0.0f, 0.0f,0);

                glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

                glPushMatrix();///We save the initial start position.
                glScalef(drawScale,drawScale,drawScale);
                glTranslatef(-drawCenter[0],-drawCenter[1],-drawCenter[2]);
                Bvh.background();

                glColor3f(0,0,1);
                glutSolidSphere(sphereSize,20,10);

                int initypos = data[1];
                float *frameData = &data[j*nchannels];//j is the frame number
                //j* nchannels skips through the array until it reaches j frame
                //times the number of channels per frame to reach the
                //start of the current frame's data in the data array
                float *specialdata = &data[dataframe*nchannels];

                root->draw(frameData,data,specialdata, j);///both "data" and
                "framedata"are passed in so that I can use the data from frame 0 as
                well as frame j. special data is the translation data for the root
                during a graph walk.
                lastframedrawn=dataframe;
                glPopMatrix();/// we restore the initial start
                ///position for the next frame. end of draw frame code

                glutSwapBuffers();
                dataframe++;

                ///now reset dataframe if we have reached the end
                if (dataframe==(nframes-1))
                {
                        dataframe=0;
                }
                stoppoint++;
                usespecialroottranslation=false;
        }

        fprintf(f,"%d %d\n",localmins[startj].j,localmins[endi].i-1);


        //finally we set startj equal to endi. Thus we are jumping from the i
        //coordinate to the j coordinate, and the cycle starts over.
        startj=endi;
        }

        fclose(f);/*And here we close the file*/
```

References

[AB] Abel, Robert and Associates, *Brilliance*, (1985)

[AF02] Arikan O., Forsyth D. A.: *Interactive motion generation from examples*. ACM Transactions on Graphics 21, 3 (2002), 483-490

[AFO03] Arikan O., Forsyth D. A., O'Brien, James F.: *Motion Synthesis from Annotations*. ACM Transactions on Graphics, Vol.: 33, No.3, (2003) pp. 402-408

[B78] Blinn, James F. *Simulation of Wrinkled Surfaces*. ACM SIGGRAPH Computer Graphics, (1978)

[GSKJ03] Gleicher, M., Joon Shin, H., Kovar, L., Jepsen, A., *Snap Together Motion*, Proceedings of the 2003 symposium on Interactive 3D graphics, (2003)

[H90] Hill JR., F.S: *Computer Graphics using OpenGL* (1990), Prentice Hall, 0023548568

[JL] Lander, Jeff, *Dagger Motion Capture File Viewer* (program)

[JM85] Jankel, A., Morton, R. *Max Headroom* (Television Show) (1985)

[KGP02] Kovar L. Gleicher M., Pighin F., *Motion Graphs*. SIGGRAPH 2002 (2002), pp. 473-482

[LA95] *Toy Story* (Animated Feature) John Lasseter

[LCF05] Lai, Y., Chenney, S, Fan, S., *Group Motion Graphs*, Eurographics/ACM SIGRAPH Symposium on Computer Animation (2005).

[ME99] Menache, Alberto. *Understanding Motion Capture for Computer Animation and Video Games*, (1999), Morgan Kaufmann, 0124906303

[MVP]. van de Panne, Michiel, Sample Motion Capture code given for course 526 at the University of British Columbia in 2005 (program). http://www.cs.ubc.ca/~van/cpsc526/

[RP04] Reitsma, P.S.A, Pollard, N.S. *Evaluating Motion Graphs for Character Navigation*, Eurographics/ACM SIGRAPH Symposium on Computer Animation (2004).

[TH87] Thalman-Magnenat, M. and Thalman, D., *The Directions of Synthetic Actors in the film Rendezvous à Montréal*, IEEE Computer Graphics and

Applications, Dec, 1987, pp.9-19. Video available at http://www.miralab.unige.ch/subpages/marilyn/VMARILYN/rendezvous.html

[WI78] Williams, L.: *The Works* (1978 -1986) (Unfinished Animated Feature)

[WP95] Witkin A., Popovic Z: *Motion Warping*. In SIGGRAPH 1995 (1995)

[WP01] Watt, A., Policarpo, F: *3D games: Real-time rendering and Software Technology*, (2001), ACM press, 0201619210

[WW92] Watt A., Watt M.: *Advanced Animation and Rendering Techniques. Theory and Practice*.(1992) Addison Wesley ISBN: 0201544121

[ZE82] Zeltzer, D., *Motor Control Techniques for Figure Animation*, IEEE Computer Graphics and Applications, 2(9), (1982)

[Zie02] Zier, D.: *Motion Capture Viewer* V1.00 (2002) (program) http://acabar.cafwap.net/~zier/anim/