IMPLEMENTATION OF MOTION GRAPHS

P-1194

PROJECT REPORT

Submitted in partial fulfillment of the requirement
for the award of the degree of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

Of

Bharathiar University

Submitted by

PREETHI .S. 0027K0191

Under the Guidance of

Dr. S. THANGASAMY PhD,
Head of the Department

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE – 641 006.

MARCH 2004

CERTIFICATE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY
COIMBATORE – 641 006.

This is to certify that the project entitled

IMPLEMENTATION OF MOTION GRAPHS

is done by

Preethi. S. 0027K0191

and submitted in partial fulfillment of the requirement for the award of the degree of the

Bachelor of Engineering
In
Computer Science and Engineering
Of
Bharathiar University, Coimbatore

[illegible]
Professor & Head of the department
(Dr. S. THANGASAMY)

[illegible]
Guide
(Dr. S. THANGASAMY)

Certified that the candidate was assessed by us in the project work

Viva voce examination held on 23-03-2004

[illegible] 23/3/'04
Internal Examiner

[illegible] 23/3/04
External Examiner

DECLARATION

I, **Preethi.S.** , (0027K0191) hereby declare that the project entitled "Implementation of Motion Graphs" is done by me and to the best of my knowledge a similar work has not been submitted to the Bharathiar University or any other institution, for fulfillment of the requirement of the course study.

This report is submitted on the partial fulfillment of the requirements for all awards of degree of Bachelor of Computer Science and Engineering of Bharathiar University.

Place: COIMBATORE

Date: 18.03.2004

Preethi. S

Preethi. S.

Acknowledgements

ACKNOWLEDGEMENTS

I would like to express my gratitude to the Principal Dr. K. K. Padmanabhan for helping me complete this project successfully.

I have immense pleasure in expressing my heartfelt thanks to the Head of the Department and my guide Dr. S. Thangasamy for his constant advice and support during the project. I am grateful to him for his guidance.

I would like to thank my project coordinator, Mrs. D. Chandrakala. Senior Lecturer, for her support during the course of my project.

I would like to express my sincere thanks to all the members of the faculty of the Department of Computer Science and Engineering for their support.

I am extremely grateful to Mrs. Alla Safonova, PhD student (Computer Graphics) at Carnegie Mellon University for helping me with the details of my project. I also wish to thank Carnegie Mellon University for extending their online resources during the course of my project.

I thank all those who have been involved directly or indirectly with my project.

Abstract

ABSTRACT

This project titled "Implementation of Motion Graphs" implements a proposal made in research paper published in the Proceedings of SIGGRAPH '02.[1] This is a method for creating realistic, controllable motion. Given a corpus of motion capture data or animations, this project automatically constructs a directed graph called a *motion graph* that encapsulates connections among the various states in the database. The motion graph consists both of pieces of original motion which is the motion capture data or the animation used, and automatically generated transitions which is created in order to create smooth transitions between the states. Simply walking through the graph, i.e. traversing the graph can generate motion.

[1]. Kovar L, Gleicher M, Pighin F, *Motion Graphs*; In the Proceedings of ACM SIGGRAPH '02.

CONTENTS

CHAPTER 1

Introduction

CHAPTER 1 INTRODUCTION

1. Introduction

Realistic human motion is an important part of media like video games and movies. More lifelike characters make for more immersive environments and more believable special effects. At the same time, realistic animation of human motion is a challenging task, as viewers have proven to be adept at discerning the subtleties of human movement and identifying inaccuracies. One common solution to this problem is motion capture. However, while motion capture is a reliable way of acquiring realistic human motion, by itself it is a technique for *reproducing* motion. Motion capture data has proven to be difficult to modify, and editing techniques are reliable only for small changes to a motion. This limits the utility of motion capture if the data on hand isn't sufficiently similar to what is desired, then often there is little that can be done other than acquire more data, a time-consuming and expensive process. This in particular is a problem for applications that require motion to be synthesized dynamically, such as interactive environments. Our goal is to retain the realism of motion capture while also giving a user the ability to generate other motions from the data available. For example, we would like to be able to ask a character to walk around a room without worrying about having a piece of motion data that contains the correct number of steps and travels in the right directions. We also need to be able to direct characters who can perform multiple actions, rather than those who are only capable of walking around. This project presents a method for synthesizing streams of motions based on a corpus of captured movement while preserving the quality of the original data. Given a set of motion capture data, we compile a structure called a *motion graph* that encodes how the captured clips may be re-assembled in different ways. The motion graph is a directed graph wherein edges contain either pieces of original motion data or automatically generated transitions. The nodes then serve as choice points where these small bits of motion join seamlessly. Because our methods automatically detect and create transitions between motions, users needn't capture motions specifically designed to connect to one another. Motion graphs transform the motion synthesis problem into one of selecting sequences of nodes, or *graph walks*.

1.1. Existing System and its Limitations

As described in the previous section, motion capture is one way of getting realistic human motion. However, manipulating a captured motion data often leads to losing of the realism of the data. Manipulating and editing of the captured motion data is difficult.

1.2. Proposed System's Advantages

Motion Graph is a way of automatically detecting and creating transitions between motions, and therefore creates new motion data from existing data. The newly created data has the realism of a captured motion data.

Thus, as new motion is created automatically this method is much faster than manually editing motion data. The time and effort that has to be devoted by animators to create motion that are 'human – like' is reduced to a very great degree. Using this system, motion data can be reused.

The above advantages give a strong motivation to pursue the project and to implement it.

CHAPTER 2

Architecture of the System

CHAPTER 2
ARCHITECTURE OF THE SYSTEM

The architecture of the system is described in this section. A block diagram representing this is shown below in Fig 1.

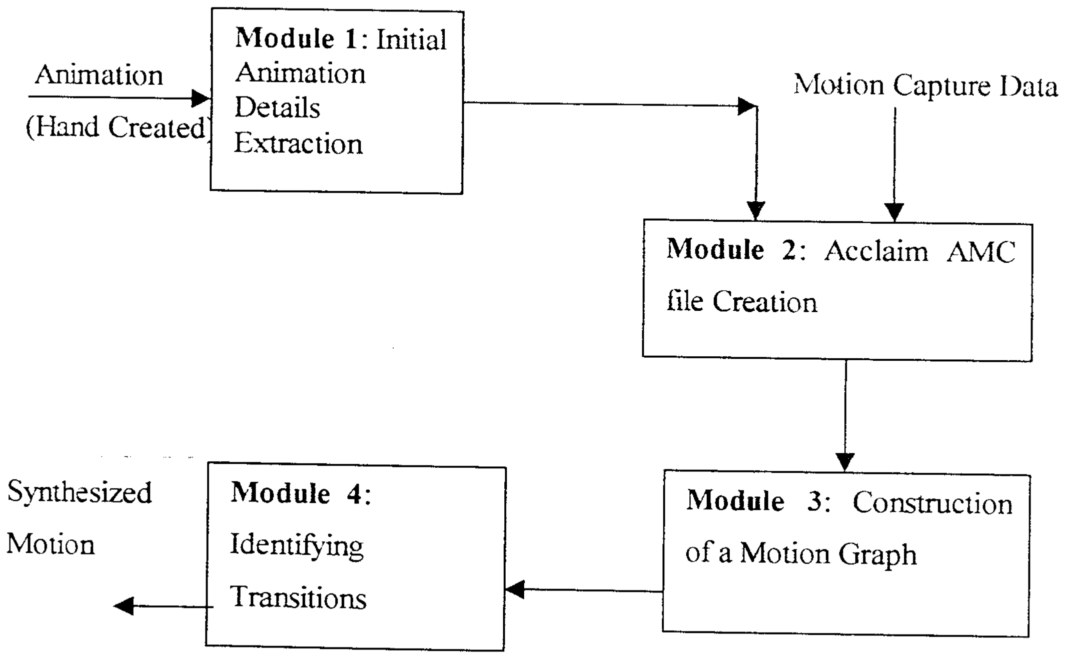
Animation
(Hand Created)
Module 1: Initial Animation Details Extraction
Motion Capture Data
Module 2: Acclaim AMC file Creation
Module 3: Construction of a Motion Graph
Module 4: Identifying Transitions
Synthesized Motion

Fig. 1. Block Diagram Representing The Architecture of The System

As seen from the figure, there are two inputs to the system, one being hand created animation and the other being motion capture data. In the case of hand created animation, an AMC file has to be created. For this purpose, the hand created animation is taken as input and only the details that are needed to create the file are extracted. This forms the core idea of module 1.

The second module involves the actual creation of the AMC file. This file contains all the details of the motion with regard to the skeleton. Once this file is created, the motion of the skeleton can be reconstructed by using this file. After this point, the AMC file alone is used for the remaining modules. The original animation does not come into the picture anymore. In case of a motion capture data, the AMC file is already available. This file can be downloaded from sites providing motion capture data.

The next block is the creation of a simple motion graph. This is the idea behind the third module. Here, the AMC file created in the previous module is used. A simple graph, whose nodes contain the details of the various joints of the skeleton for each frame and the arc represent the transition between the frames, is created. Thus, by simply traversing the graph the original motion can be obtained.

Finally, we create transition between frames such that new motion can be generated. This synthesized motion contains the original motion (from the AMC file) as well as automatically generated transitions that are created in order to create smooth transitions between frames.

Thus, the output of the system is a new synthesized motion that has the realism of the original motion, yet different from the original motion.

CHAPTER 3

System Design

CHAPTER 3
SYSTEM DESIGN

This section gives a brief overview of the modules of the project. The entire project can be divided into four modules. They are introduced below.

Module 1: Initial Animation Details Extraction

This module of the project uses the animation to extract details about the skeleton for the creation of the motion graph. The details regarding the translation and orientation of the skeleton's joints have to be extracted from the animation. Also, depending on the degrees of freedom that the joints posses, the amount of information that needs to be extracted varies. The details regarding the skeleton are given in an Acclaim Skeleton File (ASF). This file format describes animated skeletons in terms of joints (Biovision), bones (Acclaim), hierarchy and angle constraints.

Module 2: Acclaim AMC file Creation

The second module deals with the creation of an Acclaim AMC file. AMC stands for Acclaim Motion Capture file. This file contains details extracted from the animation regarding the joints for all the frames for the entire animation. The details in the AMC file represent all the details needed to produce an animation of a standard human skeleton. In the case of a motion capture data, this file is already present and files for various actions such as walk, dance etc can be got from an online database.

Module 3: Construction of a Motion Graph

This module deals with the creation of a simple motion graph that recreates the same animation as the original data. This graph does not change the contents of the original motion. The same motion is recreated as we walk through the graph. The input to this module is the AMC file containing the details of the motion for which the motion graph is to be created.

Module 4: Identifying Transitions

This module deals with identifying good transition points in order to create new motion from existing data. The difficulties of this module lies in defining a "good" transition by using an evaluation function to determine how similar are the two states between which transition has to be created.

CHAPTER 4

Programming Environment

CHAPTER 4
PROGRAMMING ENVIRONMENT

Maya software has been chosen for the implementation of this project. The hardware requirements are detailed in the next sub-section and a brief introduction to the software Maya is given in the section following that.

4.1. Hardware Requirements

Maya requires one of the following **operating systems**:

- Microsoft® Windows® XP Professional.
  Microsoft Windows 2000 Professional (Service Pack 2 or higher)
- SGI® IRIX® 6.2.15
- RedHat™ Linux® 7.3 or 8.0
- Apple® Mac® OS X 10.2.4 or higher (Maya Complete ONLY)

Maya requires one of the following **browsers**:

- Microsoft® Internet Explorer® 4.0 or higher
- Netscape® 7.0 or higher

At a minimum, Maya requires a **system** with:

- Intel® Pentium® II or higher, AMD Athlon™ processor
- 512 MB RAM
- CD-ROM Drive
- Hardware-Accelerated OpenGL® graphics card
- 3-button mouse with mouse driver software
- 450 MB of hard disk space

4.2. Introduction to Maya

4.2.1. About Alias

As the world's leading innovator of 3D graphics technology, Alias® develops award-winning software, support, custom development and training solutions for the film, video, game development, interactive media, industrial design and visualization markets. These solutions include the Maya® software product line for the entertainment industry, StudioTools™ and PortfolioWall™ software product lines for the design industry, and Alias SketchBook Pro™ a sketching, annotating and presentation software application for the Tablet PC and Wacom Tablet.

Alias software has been used in almost every film nominated by the Academy of Motion Picture Arts and Sciences in the category of Best Visual Effects since its inception. Maya played a role in eight films nominated at the 2003 Academy Awards.

Academy Award® winning Maya® 3D animation and effects software is at the forefront of technological innovation. Its development has been inspired by the film and video artists, computer game developers and design professionals who use it daily to create engaging digital imagery, animation and visual effects.

4.2.2. The Maya Family

Maya Unlimited™

This version is the world's most powerful 3D animation and visual effects software and the ultimate version of Maya®. It is possible to experience industry-leading innovations for the creation of advanced digital content. Maya Unlimited is available on Windows® XP Professional, Windows 2000 Professional, SGI IRIX® and Linux®

Maya Complete™

Maya Complete makes the foremost 3D content creation tools accessible to a broad range of computer graphics professionals in the film, broadcast, industrial design, visualization, game development and web design industries. It is the leading full 3D production solution. Maya Complete is available for Windows 2000 Professional, Windows XP Professional, Mac® OS X, SGI IRIX® and Linux operating systems.

Maya Personal Learning Edition™

Maya Personal Learning Edition™ is a special student version of Maya®, that provides free access to Maya Complete™ software for non-commercial use. This is available for Maya 5 on the Windows® 2000/XP Professional and Mac® OS X operating systems. This version is available for download from the web site www.aliaswaefront.com

4.2.3. Maya Personal Learning Edition™

Maya Personal Learning Edition™ is a special version of Maya®, that provides free access to Maya Complete™ software for non-commercial use. It is available for Maya 5 on the Windows® 2000/XP Professional and Mac® OS X operating systems. It will give 3D graphics and animation students, industry professionals, and those interested in breaking into the world of computer graphics (CG) an opportunity to explore all aspects of the award winning Maya Complete™ software in a non-commercial capacity. This version has been used to develop the project.

Maya Personal Learning Edition 5 Offers:

- A new hardware rendering option using the power of next-generation graphics cards. Get near software-quality images at often dramatically faster speeds.
- A unified rendering workflow for easy and consistent access through a common interface.
- Animation enhancements to constraints, forward/inverse kinematics and ghosting for added flexibility.
- Maya Paint Effects™ to polygon conversion for a whole new range of looks plus more editing and output options.
- An improved polygon reduction method for making lightweight game models.

CHAPTER 5

Detailed Design

CHAPTER 5
DETAILED DESIGN

This section gives a detailed description of the modules of the project. As seen before, the project can be divided into four modules. The module details are given below.

Module 1: Initial Animation Details Extraction

One of the inputs for this project is from hand created animations, the other being motion capture data. This module of the project uses the animation to extract details about the skeleton for the creation of the motion graph.

What are skeletons?

Skeletons in a 3D character animation have a direct correlation to a human skeleton: they consist of articulated joints and bones, and they can be used as a controlling mechanism to deform attached mesh data via skinning. After a skeleton is created, it is bound with a character's surface to cause the surface to move with the skeleton during animation. Binding is also called skinning, and a character's surface after binding is called skin. Near joints, the skin bulges or indents when you rotate the joints. This is useful for animating elbows, shoulders, necks, and so on.

Refer to Fig 2. The skeleton is actually just composed of nodes in 3D space (or "grouping nodes" as they are also often called). Parenting the nodes together creates an explicit hierarchy, and the transformations on the nodes define the rotation and offset of each node from its parent node. The location of each node coincides with a "joint" and the distance between two child-parent nodes defines the length of the bone. A skeleton is defined as a series of connected joints. A joint also has rotational and positional constraints.

Some 3D programs are "joint based" and others "bones based". Alias Maya is "joint based" which means that the bone is visualized implicitly between two joint nodes (two null nodes). Thus, you always need at least two joints to define a bone. Other programs, such as 3D MAX and Lightwave, are "bone based" which means that a bone is visualized based on a starting location, direction and bone length (a child joint node is not necessary in these programs for a bone to become visible). File formats such as Acclaim also work in terms of bones and not joints

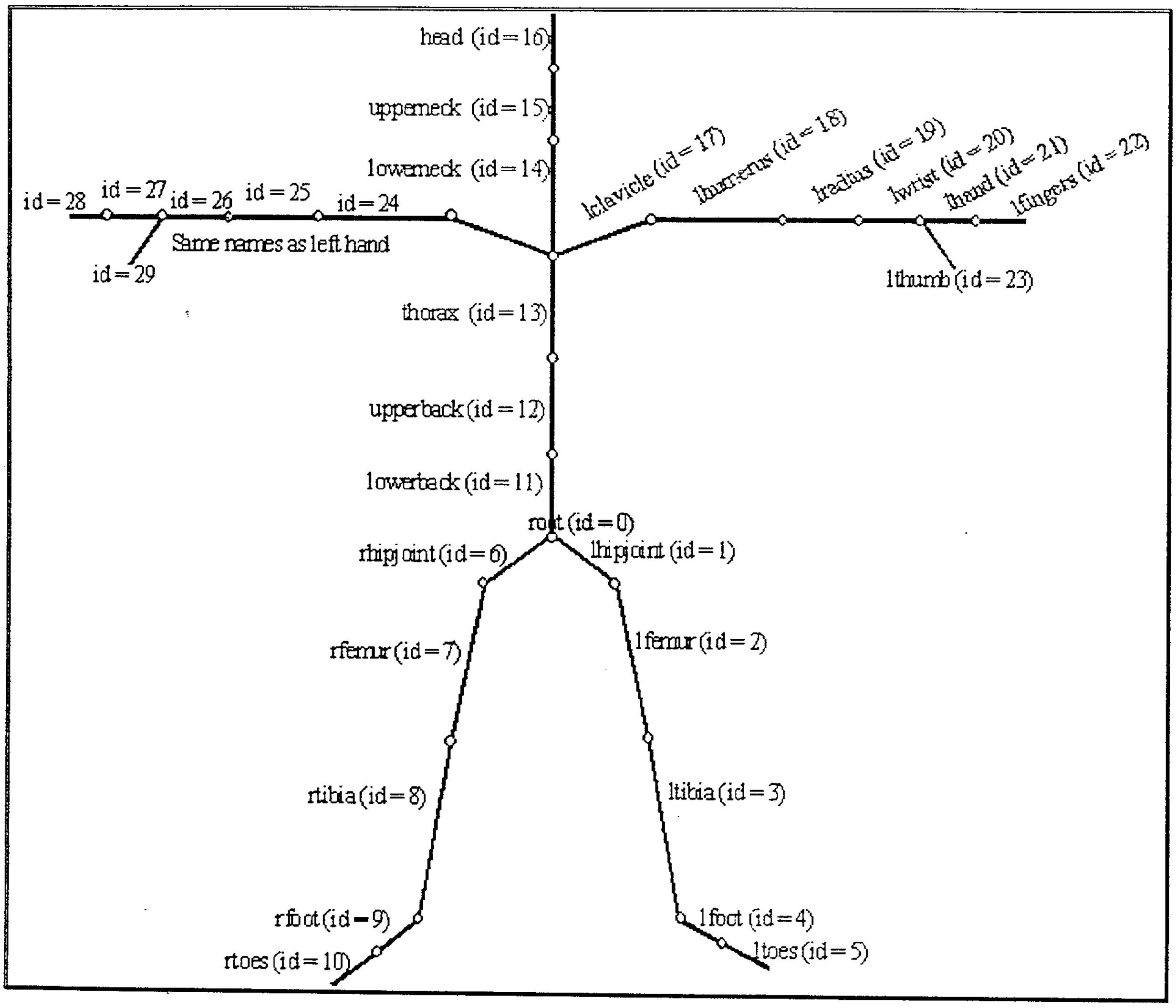

Fig 2. Skeleton hierarchy as defined in ASF file (length of the bones is not up to scale).

Acclaim ASF File Format

The details regarding the skeleton are given in an Acclaim Skeleton File (ASF). This file format describes animated skeletons in terms of joints, bones, hierarchy and angle constraints.

Acclaim is a game company that has been doing research into motion capture for games for many years. They developed their own methods for creating skeleton motion from optical tracker data and subsequently devised a file format, actually two files, for storing the skeleton data. Later they put the format description in the public domain for anyone to use. Oxford Metrics, makers of the Vicon motion capture system, elected to use the Acclaim format as the output format of their software.

The Acclaim format is made up of two files, a skeleton file and a motion file. This was done knowing that most of the time a single skeleton works for many different motions and rather than storing the same skeleton in each of the motion files it should be stored just once in another file. The skeleton file is the ASF file (Acclaim Skeleton File). The motion file is the AMC file (Acclaim Motion Capture data).

Parsing the ASF file

In the ASF file a base pose is defined for the skeleton that is the starting point for the motion data. Each segment has information regarding the way the segment is to be drawn as well as information that can be used for physical dynamics programs, inverse kinematic programs or skinning programs. One of the peculiar features of the ASF file is the requirement that there be no gaps in the skeleton. No child can have a non-zero offset from the end of the parent segment. This has the effect of creating more skeletal segments than are usually found in other file formats. A limitation of the ASF definition is that only one root exists in the scene, this doesn't prevent a file from cleverly containing two skeletons attached to the root but it does make such a construction clumsy. A sample ASF file is given below followed by a detailed description of the contents of the file and how to interpret the file.

Sample ASF File

Examine the example file "Style4.asf'. The exact contents of the file are given below.

```
# AST/ASF file generated using VICON BodyLanguage
# --------------------------------------------
:version 1.10
:name VICON
:units
  mass 1.0
  length 0.45
  angle deg
:documentation
   .ast/.asf automatically generated from VICON data using
   VICON BodyBuilder and BodyLanguage model FoxedUp or BRILLIANT.MOD
:root
   order TX TY TZ RX RY RZ
   axis XYZ
   position 0 0 0
   orientation 0 0 0
:bonedata
  begin
    id 1
    name lhipjoint
    direction 0.661853 -0.702389 0.261917
    length 2.64193
    axis 0 0 0  XYZ
  end
  begin
    id 2
    name lfemur
    direction 0.34202 -0.939693 0
    length 7.01722
    axis 0 0 20  XYZ
   dof rx ry rz
   limits (-160.0 20.0)
        (-70.0 70.0)
        (-60.0 70.0)
  end
  begin
    id 3
    name ltibia
    direction 0.34202 -0.939693 0
    length 8.25008
    axis 0 0 20   XYZ
   dof rx
   limits (-10.0 170.0)
  end
  begin
    id 4
    name lfoot
    direction 0.0709916 -0.195048 0.978221
    length 2.31941
    axis -90 7.62852e-016 20   XYZ
   dof rx rz
   limits (-45.0 90.0)
```

```
        (-70.0 20.0)
  end
  begin
    id 5
    name ltoes
    direction 1.53586e-011 -4.22017e-011 1
    length 1.16486
    axis -90 7.62852e-016 20   XYZ
   dof rx
   limits (-90.0 20.0)
  end
  begin
    id 6
    name rhipjoint
    direction -0.638185 -0.721362 0.268992
    length 2.57244
    axis 0 0 0  XYZ
  end
  begin
    id 7
    name rfemur
    direction -0.34202 -0.939693 0
    length 7.07017
    axis 0 0 -20   XYZ
   dof rx ry rz
   limits (-160.0 20.0)
        (-70.0 70.0)
        (-70.0 60.0)
  end
  begin
    id 8
    name rtibia
    direction -0.34202 -0.939693 0
    length 8.20386
    axis 0 0 -20   XYZ
   dof rx
   limits (-10.0 170.0)
  end
  begin
    id 9
    name rfoot
    direction -0.0822429 -0.225961 0.970658
    length 2.32215
    axis -90 -7.62852e-016 -20   XYZ
   dof rx rz
   limits (-45.0 90.0)
        (-20.0 70.0)
  end
  begin
    id 10
    name rtoes
    direction -1.53597e-011 -4.21689e-011 1
    length 1.16333
    axis -90 -7.62852e-016 -20   XYZ
   dof rx
   limits (-90.0 20.0)
  end
  begin
```

```
      id 11
      name lowerback
      direction -0.0131557 0.998589 -0.0514484
      length 2.14993
      axis 0 0 0   XYZ
     dof rx ry rz
     limits (-20.0 45.0)
           (-30.0 30.0)
           (-30.0 30.0)
   end
   begin
     id 12
     name upperback
     direction -0.000776481 0.999993 0.00352685
     length 2.15301
     axis 0 0 0   XYZ
    dof rx ry rz
    limits (-20.0 45.0)
          (-30.0 30.0)
          (-30.0 30.0)
  end
  begin
    id 13
    name thorax
    direction 0.00438186 0.999609 0.0276143
    length 2.15424
    axis 0 0 0   XYZ
   dof rx ry rz
   limits (-20.0 45.0)
         (-30.0 30.0)
         (-30.0 30.0)
 end
 begin
   id 14
   name lowerneck
   direction -0.0488006 0.998094 0.0377807
   length 1.90481
   axis 0 0 0   XYZ
  dof rx ry rz
  limits (-20.0 45.0)
        (-30.0 30.0)
        (-30.0 30.0)
end
begin
  id 15
  name upperneck
  direction 0.0443769 0.990702 -0.12861
  length 1.91305
  axis 0 0 0   XYZ
 dof rx ry rz
 limits (-20.0 45.0)
       (-30.0 30.0)
       (-30.0 30.0)
end
begin
  id 16
  name head
  direction 0.0165599 0.998647 -0.0492982
```

```
    length 1.92369
    axis 0 0 0   XYZ
   dof rx ry rz
   limits (-20.0 45.0)
          (-30.0 30.0)
          (-30.0 30.0)
  end
  begin
    id 17
    name lclavicle
    direction 0.911936 0.410197 -0.0105506
    length 3.52622
    axis 0 0 0   XYZ
   dof ry rz
   limits (-20.0 10.0)
          (0.0 20.0)
  end
  begin
    id 18
    name lhumerus
    direction 1 -4.48974e-011 -2.48591e-028
    length 5.47651
    axis 180 -30 -90   XYZ
   dof rx ry rz
   limits (-60.0 90.0)
          (-90.0 90.0)
          (-90.0 90.0)
  end
  begin
    id 19
    name lradius
    direction 1 -4.48938e-011 -4.40099e-027
    length 3.09058
    axis 180 -30 -90   XYZ
   dof rx
   limits (-10.0 170.0)
  end
  begin
    id 20
    name lwrist
    direction 1 -4.49042e-011 -2.99852e-027
    length 1.54529
    axis 1.95339e-015 90 90   XYZ
   dof ry
   limits (-180.0 0.0)
 end
  begin
    id 21
    name lhand
    direction 1 -4.48815e-011 -5.98193e-027
    length 0.771611
    axis 3.82337e-015 90 90   XYZ
   dof rx rz
   limits (-90.0 90.0)
          (-45.0 45.0)
  end
  begin
    id 22
```

```
     name lfingers
     direction 1 -4.49034e-011 -1.1993e-026
     length 0.622093
     axis 7.64668e-015 90 90   XYZ
    dof rx
    limits (0.0 90.0)
  end
  begin
     id 23
     name lthumb
     direction 0.707107 -6.34867e-011 0.707107
     length 0.893204
     axis -90 45 -1.62303e-014   XYZ
    dof rx rz
    limits (-45.0 45.0)
           (-45.0 45.0)
  end
  begin
     id 24
     name rclavicle
     direction -0.93442 0.355634 -0.0195692
     length 3.62463
     axis 0 0 0   XYZ
    dof ry rz
    limits (-10.0 20.0)
           (-20.0 0.0)
  end
  begin
     id 25
     name rhumerus
     direction -1 -4.48957e-011 -1.19147e-027
     length 5.14182
     axis 180 30 90   XYZ
    dof rx ry rz
    limits (-90.0 60.0)
           (-90.0 90.0)
           (-90.0 90.0)
  end
  begin
     id 26
     name rradius
     direction -1 -4.48952e-011 -3.81062e-027
     length 3.35565
     axis 180 30 90   XYZ
    dof rx
    limits (-10.0 170.0)
  end
  begin
     id 27
     name rwrist
     direction -1 -4.48973e-011 -2.99621e-027
     length 1.67782
     axis 1.95339e-015 -90 -90   XYZ
    dof ry
    limits (-180.0 0.0)
  end
  begin
     id 28
```

```
      name rhand
      direction -1 -4.48895e-011 -5.98727e-027
      length 0.801042
      axis 3.82337e-015 -90 -90   XYZ
     dof rx rz
     limits (-90.0 90.0)
          (-45.0 45.0)
   end
   begin
     id 29
     name rfingers
     direction -1 -4.49065e-011 -1.19972e-026
     length 0.645821
     axis 7.64668e-015 -90 -90   XYZ
    dof rx
    limits (0.0 90.0)
   end
   begin
     id 30
     name rthumb
     direction -0.707107 -6.34957e-011 0.707107
     length 0.927273
     axis -90 -45 1.62303e-014   XYZ
    dof rx rz
    limits (-45.0 45.0)
          (-45.0 45.0)
  end
:hierarchy
  begin
    root lhipjoint rhipjoint lowerback
    lhipjoint lfemur
    lfemur ltibia
    ltibia lfoot
    lfoot ltoes
    rhipjoint rfemur
    rfemur rtibia
    rtibia rfoot
    rfoot rtoes
    lowerback upperback
    upperback thorax
    thorax lowerneck lclavicle rclavicle
    lowerneck upperneck
    upperneck head
    lclavicle lhumerus
    lhumerus lradius
    lradius lwrist
    lwrist lhand lthumb
    lhand lfingers
    rclavicle rhumerus
    rhumerus rradius
    rradius rwrist
    rwrist rhand rthumb
    rhand rfingers
  end
```

Here you will see that keywords in the file all start with a colon ":". Keywords will either set global values or they will indicate the beginning of a section of data.

The **":version"** keyword indicates the version of the skeleton definition. This document is for version 1.10.

The **":name"** keyword allows the skeleton to be named something other than the file name.

The **":units"** keyword denotes a section that defines the units to be used for various types of data. It also defines default values for other parameter. Any number of specifications may be found here, the use of these values are often program specific. Ideally you should store these values and then write them out again if you make modifications to a file. If you intend to just read the motion data in then you can ignore those values that don't interest you. In this section you will find the units used for angles and sometimes the default values for mass and length of segments.

The **":documentation"** section allows for the storage of documentation information that will persist from one file creation to the next. Simple comment information in the file is not guaranteed to be retained if the file is read into memory than saved to another file, possibly with modifications.

The **":root"** section defines a special segment of the scene (well, it's special to the way the file format is defined, you can really treat this just like any other segment in all other ways). This is the root segment of the skeleton hierarchy. It is much like all the other segments but doesn't contain direction and length information. The **"axis"** keyword in the root section defines the rotation order of the root object. The **"order"** keyword specifies the channels of motion that are applied to the root and in what order they will appear in the AMC file. The **"position"** and **"orientation"** keywords are each followed by a triplet of numbers indicating the starting position and orientation of the root. These are typically, but not always, zero.

The **":bonedata"** section contains a description of each of the segments in the hierarchy. These descriptions are for just the segments. The hierarchy section, which comes next, will describe the parenting organization of the segments. The segment definition is bracketed by a **"begin"** and **"end"** pair of keywords (note the lack of a colon in each keyword). Within the segment definition you will find:

- **"id"** This is a number that provides a unique id for the segment. This really isn't necessary since each segment is also named and the name is used for both the hierarchy section and in the AMC file.
- **"name"** This gives the name of the segment. Each segment must have a unique name. Often you will see segments with similar names such as "hips" and "hips1". The segments that have numbers at the end will usually be the children of the segment with the same name but no number. The segments with the numbers are usually there simply to fill the gap between the parent segment and a child segment. Often the segments with a number at the end will not have any motion capture data. You don't treat these in any special way; this is just noted as an **interesting feature**.
- **"direction"** This is the direction of the segment. This defines how the segment should be drawn and it also defines the direction from the parent to the child segment(s). The direction and length of a segment determine the offset of the child from the parent, if there is a child of the segment.
- **"length"** The length of the segment. With the direction value gives the information needed for drawing the segment.
- **"axis"** This gives an axis of rotation for the segment. By specifying this as a separate value the motion data can be independent of the drawing and hierarchy information. This is particularly useful for those applications that might provide motion-editing tools that are sensitive to gimbal lock.
- **"dof"** This stands for "Degrees of Freedom" and specifies the number of motion channels and in what order they appear in the AMC file. If the dof keyword doesn't appear then the segment doesn't get any motion data. No translation channels will ever appear here only rotation channels and sometimes a length channel.
- **"limits"** This specification provides for putting limits on each of the channels in the dof specification. For each channel that appears there will be a pair of numbers inside parenthesis indicating the minimum and maximum allowed value for that

channel. This information is not used for interpreting the motion data, this is useful only for those applications that might apply motion-editing functions that put limits on rotation. This also does not say that the given numbers might limit the data in the AMC file.

The next figure, Fig 3 shows rotation axes and names of some bones. You can view the local coordinate system for each bone by entering its bone index, which is defined in .ASF file and shown in Fig 2.

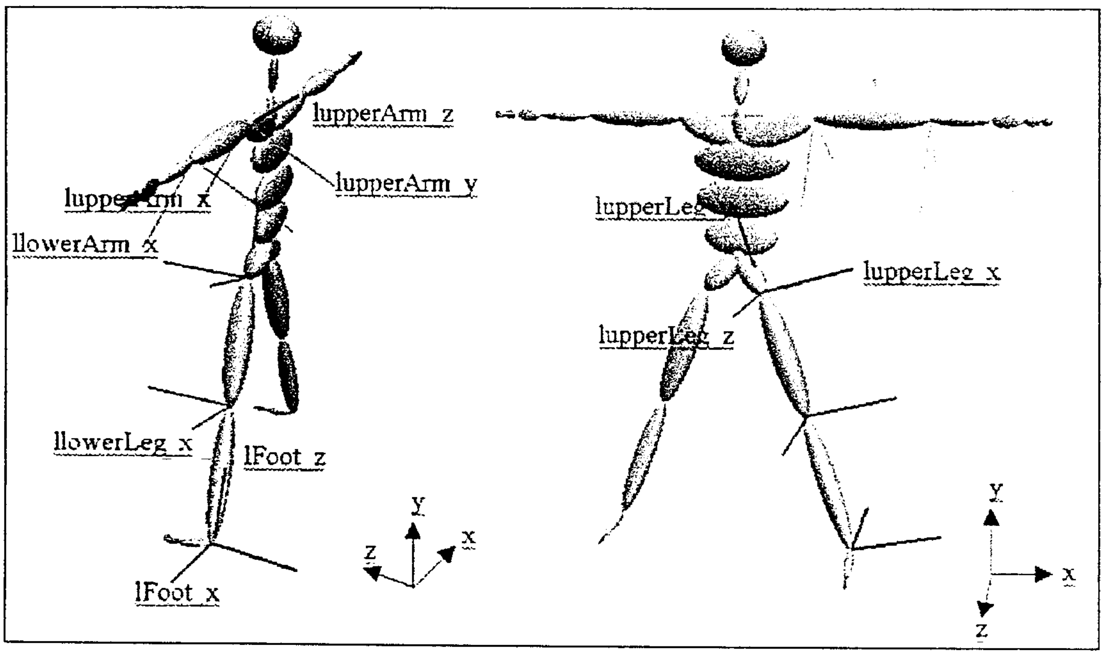

Fig 3. Shows local coordinate system for some of the bones

How To Extract The Appropriate Details?

The animation will have a lot of details, say the skeleton, character's skin, environment's details, lighting, particles, camera, constraints and the like. This module deals with extracting just the information relating to the skeleton.

In order to do this, this module exploits the **Hypergraph** structure present in Maya.

What is a Hypergraph?

The Hypergraph shows a graphical relationship between components of a scene. You can display two kinds of graphs in the Hypergraph: the scene hierarchy or dependency graph.

The *scene hierarchy* as shown in Fig 4 shows the ordered arrangement of objects, lights, cameras, and other items that make up a scene. It has features and visual aids for working with the hierarchy of scene components.

Here's an example scene hierarchy:

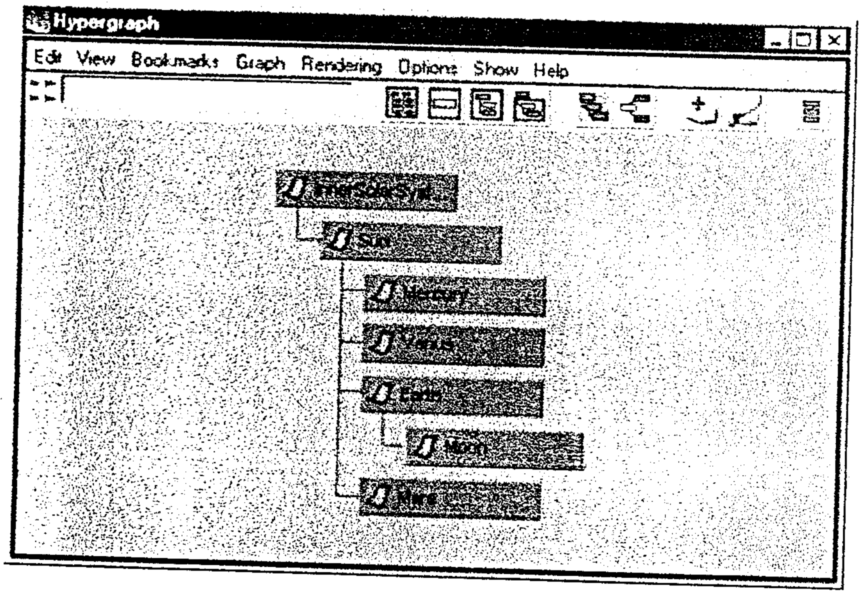

Fig 4. Scene Hierarchy

Parent

This is an object or other item that controls attributes of one or more children. A parent can also be the child of another parent. In the figure, InnerSolarSystem is a parent of Sun. Sun is a parent of Mercury, Venus, Earth, and Mars. Earth is a parent of Moon.

Child

This is an object having attributes controlled by its parent. A child can be the parent of other children. A child in the graph is connected to its parent by an indented right angle line. Sun is a child of InnerSolarSystem. Mercury, Venus, Earth, and Mars are children of Sun. Moon is a child of Earth.

Node

This is a parent, child, or independent item. This refers generally to any box in the graph. InnerSolarSystem, Sun, Moon, and all other boxes in the graph are nodes.

Subnode

Any node below another node in the hierarchy is called a subnode. The subnodes of InnerSolarSystem are Sun, Mercury, Venus, Earth, Moon, and Mars. The subnodes of Sun are Mercury, Venus, Earth, Moon, and Mars. The subnode of Earth is Moon. Moon and Mars have no subnode.

Branch

All nodes along a path from a parent to child are called branches. A branch from Sun to Moon includes Sun, Earth, and Moon.

Hierarchy

The arrangement of all connected nodes that make up a scene or object is called a hierarchy. The scene hierarchy is made of all nodes in the figure. The Earth hierarchy consists of Earth and Moon.

Transform node

A node that contains an object's transformation attributes—values for it's translate, rotation, scale, and so on is called a transform node. It also holds information on parent-child relationships it has with other nodes. InnerSolarSystem, Sun, Moon, and all other boxes shown in the example are transform nodes.

Shape node

A shape node holds an object's geometry attributes or attributes other than the object's transform node attributes. Shape nodes do not appear in the scene hierarchy by default.

A *dependency graph* shows the architectural connections between Maya entities that input and output data. For example, it shows connections between shading group elements that create an object's material appearance.

Suppose you create a NURBS sphere, then use the Hypershade to create and assign a Phong shading group to it. Next you use the Hypershade to create a 2D checker texture and assign it to the Phong node.

The Hypershade displays the following contents (Fig 5):

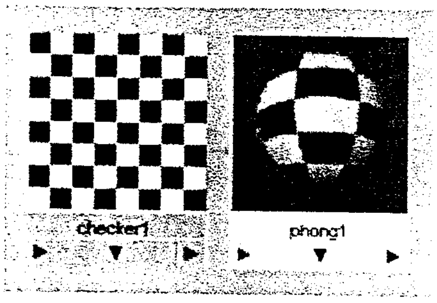

Fig 5. Hypershade

The following dependency graph appears when you select Rendering > Show Shading Groups in the Hypergraph (Fig 6).

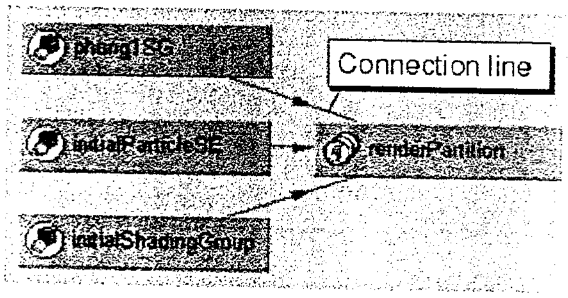

Fig 6. Dependency Graph

The connection lines between nodes show connection direction. The connection line originates at a node that outputs data, and the line points to a node receiving the data as input.

The preceding figure shows that the flow of output goes from the phong1SG shading group to the renderPartition.

Although you can see most of the same nodes in the Hypershade, the dependency graph shows the nodes in a flow diagram. This makes it easy to see the connections between the nodes that make up a shading group.

DAG Hierarchy

In Maya, such things as the position, orientation, and scale of geometry is defined by a *directed acyclic graph*, the DAG. The DAG is composed of two types of DAG nodes, transforms and shapes.

- Transform nodes maintain transformation information (position, rotation, scale, etc.) as well as parenting information. For example, if you model a hand, you would like to apply a single transformation to rotate the palm and fingers, rather than rotating each individually—in this case the palm and fingers would share a common parent transformation node.

- Shape nodes reference geometry and do not provide parenting or transformation information.

In the simplest case, the DAG describes how an instance of an object is constructed from a piece of geometry. For example, when you create a sphere, you create both a shape node (the sphere itself) and a transformation node that allows you to specify the sphere's position, its scaling, or its rotation. The transformation node would have the shape node as its child.

Nodes

Transform nodes may have multiple child nodes, in which case the child nodes are considered to be "grouped" beneath the transformation node. Node grouping allows them to share transformation information and be treated as a unit.

Instancing

Transform nodes and shape nodes may also have multiple parent nodes, in which case the nodes are considered to be "instanced". Instancing is a useful means for reducing the amount of geometry in a model. For example, if you modeled a tree, you could create a thousand unique leaves to populate the tree. This would make for a very data heavy model, since each leaf would have it's own transformation nodes, shape nodes, and NURBS or polygon data. Alternatively you could create a single leaf and instance it a thousand times to create a thousand identical leaves that could be positioned independently around the branches of the tree. This way the shape node and NURBS or polygon data would be shared and you would only require a thousand transform nodes.

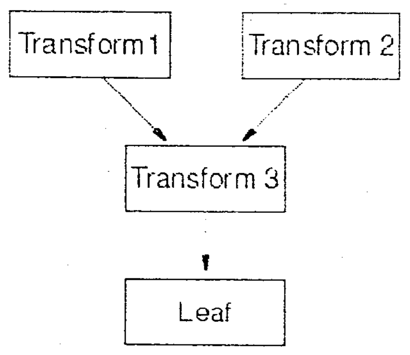

Fig 7. DAG Hierarchy showing Instancing

The DAG hierarchy shown in Fig 7 has three transform nodes (Transform1, Transform2, Transform3) and one shape node (Leaf). This DAG hierarchy would cause two leaves to be displayed since Transform3 and therefore the Leaf is instanced (it has two parents).

Transforms and shapes

A DAG node is simply an entity in the DAG. It may have and know about parents, siblings, and children, but it does not necessarily know about transformations or geometry. Transforms and Shapes are two types of nodes that are derived from a DAG node. A transform node is a type of DAG node that handles transformations (translate, rotate, and scale), while a shape node is a type of DAG node that handles geometry. A shape node does not maintain transformation information, and geometry cannot be hung below a transform node. This means that any piece of geometry requires two DAG nodes above it, a shape node immediately above it, and a transform node above the shape node. For example:

MFnDagNode has methods for determining the number of parents, and the parents of a node.

MFnTransform, the function set for operating on transform nodes (derived from MFnDagNode), has methods to get and set transformation, such as rotation, translation, or scale.

MFnNurbsSurface, one of many types of function sets which operate on the many types of shape nodes (also derived from MFnDagNode, but not derived from MFnTransform), has methods to get and set the CVs of the surface, etc.

DAG paths

A path through the DAG is a set of nodes that uniquely identifies the location of a particular node or instance of a node in the graph. The path represents a graph ancestry beginning with the root node of the graph and containing, in succession, a particular child of the root node followed by a particular child of this child, etc., down to the node identified by the path. For instanced nodes, there are multiple paths that lead from the root node to the instanced node, one path for each instance. Paths are displayed in Maya by naming each node in the path starting with the root node and separated by the vertical line character, "|".

It is important to note that because the DAG path represents how a shape is inserted into the scene, a DAG path must be used when attempting any world space operation via the API. If one simply gets an "MObject" handle to a node and asks for the world space position of a component of that node, the API operation will fail. This is because without the DAG path Maya has no idea where in world space the object is. In fact, in the case of instanced objects, there can be multiple answers to that question, and only the DAG path will uniquely identify the particular instance of the node. Almost all of the classes that contain methods that will return an MObject for a node also contain methods that will return DAG paths so you can get an MDagPath handle to the desired node. As well, all the MFn classes can be constructed with either an MObject or an MDagPath. If an MObject is used, all world space operations attempted using methods of the MFn class will fail, if an MDagPath is used, they will succeed.

Thus by traversing the DAG structure, all the joints of the skeleton are accessed. Associated with these nodes are details including the degrees of freedom that the joint possesses etc. Using the appropriate objects, these details are accessed and stored. From the details got the AMC file is created.

Module 2: Acclaim AMC file

The second module deals with the creation of an Acclaim AMC file. This file contains details extracted from the animation regarding the joints of the skeleton for all the frames for the entire animation. In the case of a motion capture data, this file is already present and files for various actions such as walk, dance etc can be got from an online database.

AMC File Format

The format of an .amc file for n joints and m frames is as shown below.

```
Frame# 1
Joint name1    translation details    orientation details
...            ...                    ...
...            ...                    ...
Joint name n   translation details    orientation details


...
...
Frame# m
Joint name1    translation details    orientation details
...            ...                    ...
...            ...                    ...
Joint name n   translation details    orientation details
```

Parsing the AMC file

The AMC file contains the motion data for a skeleton defined by an ASF file. The motion data is given a sample at a time. As can be seen from the sample AMC file given in the next section, each sample consists of a number of lines, a segment per line, containing the data. The start of a sample is denoted by the sample number alone on a line. For each segment the segment name appears followed by the numbers in the order specified by the Degree of Freedom (dof) keyword in the ASF file.

The frame number for which the details are being recorded is shown alone on a line. In the next few lines, the details of the joints of the skeleton are recorded. This detail contains the name of the joint, then the translation and orientation value as present at that frame. In case of the root joint, all three translation details (x, y, z) and all three orientation details (x, y, z) are recorded. For all other joints only the orientation details need be recorded. Looking at the sample AMC file, we observe that some joints have three values recorded, some two, some just one value while some joints are not recorded at all. The orientation details recorded depends on the degrees of freedom the joint possesses. If the joint possesses all three degrees of freedom, i.e., rotation in the x, y and z axes, then all three orientation values are recorded in the same order. (Example, *lowerback 7.86573 1.57272 6.3571*) For joints that have two degrees of freedom, only the values for which a degree of freedom exists will be recorded. (Example, *rhand -25.9429 18.5743*). Similarly for joints for a single degree of freedom only one value will be recorded (Example *rradius 46.2257*), and for joints that have no degrees of freedom, the joints are not recorded at all (Example, the hip joint). The above procedure is repeated for all the joints of the skeleton and for all the frames of the animation.

Sample AMC File

A sample AMC file is shown below for a motion with 4 frames.

```
#!OML:ASF F:\VICON\USERDATA\INSTALL\rory3\rory3.ASF
:FULLY-SPECIFIED
:DEGREES
1
root -14.8475 17.9014 10.0609 -0.613636 -32.7538 1.56106
lowerback 12.48 1.72796 0.826228
upperback 0.052032 2.38727 -0.273532
thorax -6.90601 1.12814 -1.18831
lowerneck -23.3796 0.360899 -8.68613
upperneck 35.4884 3.35267 2.06298
head 16.5761 0.394255 3.08775
rclavicle -3.16316e-014 8.94531e-015
rhumerus -35.4714 -13.8849 -82.1702
rradius 32.2782
rwrist 6.33516
rhand -12.6615 8.04265
rfingers 7.12502
rthumb 13.4195 -21.5051
lclavicle -3.16316e-014 8.94531e-015
lhumerus -28.7223 5.05714 85.111
lradius 33.3076
lwrist -5.34301
lhand -18.9786 26.2885
lfingers 7.12502
lthumb 7.32577 56.1417
rfemur -14.1915 -2.3224 24.5866
rtibia 31.1082
rfoot -12.0207 -7.35503
rtoes 8.62845
lfemur -13.9468 -3.6374 -20.5688
ltibia 26.6938
lfoot -6.2409 -3.27588
ltoes -1.67601
2
root -14.8468 17.9056 10.0229 -0.486841 -32.2878 1.47636
lowerback 12.6066 1.82474 1.11913
upperback 0.0944781 2.55656 -0.305776
thorax -6.91371 1.20006 -1.42148
lowerneck -23.2661 0.906628 -8.54134
upperneck 35.7578 4.22055 1.81451
head 16.6324 0.72172 3.13157
rclavicle 9.9939e-014 -8.34896e-015
rhumerus -35.5353 -14.0164 -82.1357
rradius 32.3942
rwrist 6.41686
rhand -12.6701 8.0652
rfingers 7.12502
rthumb 13.4112 -21.4831
lclavicle 9.9939e-014 -8.34896e-015
lhumerus -28.4171 5.10255 85.1426
```

```
lradius 33.4096
lwrist -5.23786
lhand -18.8427 25.9772
lfingers 7.12502
lthumb 7.45697 55.8258
rfemur -14.3636 -2.65457 24.71
rtibia 31.028
rfoot -11.8616 -7.34487
rtoes 8.74696
lfemur -14.3863 -3.95758 -20.3548
ltibia 27.0324
lfoot -5.94493 -3.55462
ltoes -2.48553
3
root -14.8468 17.9093 9.98408 -0.501316 -31.7481 1.65807
lowerback 12.683 1.93955 1.19253
upperback 0.175682 2.71224 -0.405437
thorax -6.85722 1.27271 -1.59701
lowerneck -23.2989 1.49623 -8.34837
upperneck 36.1812 5.1199 1.67584
head 16.802 1.07456 3.23143
rclavicle 1.66234e-014 -1.78906e-014
rhumerus -35.5536 -14.2812 -82.0663
rradius 32.4685
rwrist 6.51396
rhand -12.6688 8.52019
rfingers 7.12502
rthumb 13.4125 -21.028
lclavicle 1.66234e-014 -1.78906e-014
lhumerus -28.1373 5.06207 85.2346
lradius 33.6466
lwrist -5.02418
lhand -18.8636 26.2906
lfingers 7.12502
lthumb 7.43681 56.1399
rfemur -14.5056 -2.91806 24.5704
rtibia 30.9363
rfoot -11.7089 -7.41919
rtoes 8.64774
lfemur -14.8078 -4.51334 -20.3923
ltibia 27.3361
lfoot -6.03576 -3.65021
ltoes -5.11759
4
root -14.8476 17.9189 9.9427 -0.448767 -31.2461 1.70527
lowerback 12.7997 2.02315 1.30271
upperback 0.263413 2.83765 -0.42782
thorax -6.81865 1.33052 -1.70782
lowerneck -23.472 2.13653 -8.16778
upperneck 36.7485 6.12579 1.55514
head 17.0795 1.45791 3.37414
rclavicle -4.64659e-014 7.95139e-015
rhumerus -35.4985 -14.3793 -82.003
```

```
rradius 32.4903
rwrist 6.56498
rhand -12.5413 9.00745
rfingers 7.12502
rthumb 13.5353 -20.5326
lclavicle -4.64659e-014 7.95139e-015
lhumerus -27.9669 5.18733·85.3046
lradius 34.0504
lwrist -4.66987
lhand -18.9805 26.1724
lfingers 7.12502
lthumb 7.32395 56.0256
rfemur -14.6639 -3.58318 24.5561
rtibia 30.8361
rfoot -11.5622 -7.10357
rtoes 8.19772
lfemur -15.2574 -4.80797 -20.2977
ltibia 27.69
lfoot -6.33027 -3.897
ltoes -7.5911
5
root -14.8488 17.9236 9.90465 -0.333738 -30.8493 1.6438
lowerback 12.887 2.13205 1.50826
upperback 0.330889 3.0135 -0.413496
thorax -6.77974 1.41061 -1.84205
lowerneck -23.2862 2.82007 -7.96211
upperneck 37.0147 7.206 1.25527
head 17.1256 1.86945 3.42347
rclavicle -2.70347e-014 1.17283e-014
rhumerus -35.4889 -14.392 -81.9354
rradius 32.5354
rwrist 6.58573
rhand -12.4436 9.13891
rfingers 7.12502
rthumb 13.6295 -20.3949
lclavicle -2.70347e-014 1.17283e-014
lhumerus -27.8724 5.44068 85.348
lradius 34.4564
lwrist -4.44633
lhand -18.7222 26.034
lfingers 7.12502
lthumb 7.57334 55.8784
rfemur -14.8163 -3.90687 24.6493
rtibia 30.7579
rfoot -11.379 -7.03465
rtoes 8.27206
lfemur -15.6921 -5.00245 -20.0786
ltibia 28.0788
lfoot -6.55424 -4.22637
ltoes -8.51631
```

Module 3: Construction of a Motion Graph

This module deals with the creation of a simple motion graph that recreates the same animation as the original data. This graph does not change the contents of the original motion. The same motion is recreated as we walk through the graph.

The input of this module is the .amc file created in the previous module. Maya is not used anymore in this project. The .amc file is read. For each frame the translation and rotation for each joint is taken from the .amc file. Depending on what is read, the skeleton is positioned at the specified location and in the specified angle. This is done till all the frames present in the .amc file are read. Thus, the nodes of the graph are the individual frames from the AMC file and a simple graph is created using the frames present. The graph created is shown below in Fig 8.

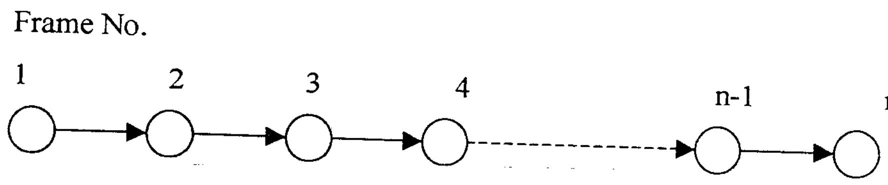

Fig 8. Simple Motion Graph.

Interface Structure

This module has an interface through which loading of any skeleton and any motion is possible. This interface has the basic functionalities of being able to open any file from the list of directories present in the system. The advanced features that this interface supports include camera features like translation of the scene, zooming and moving the camera at any angle and position; a time slider that has keys for pause, play, stop, loop back and record; and a feature for saving individual frames as well as the entire animation sequence in the form of jpeg files. All the above features mention have been implemented using VC++.

Difficulties

The difficulties that were encountered while implementing this module were multifold. One of the main difficulties was to create the skeleton. A lot of effort was expended in actually rendering the skeleton on the screen. Other difficulty was in storing the state information of all the frames. The camera features that have been implemented also proved to be non-trivial. Mouse movements had to be captured and depending on the mouse button clicked and the relative position of the mouse movements, the camera had to be repositioned. As the camera moves, the view of the skeleton also changes. This implementation also proved to be non-trivial. The implementation of the time slide was also difficult. All other features were relatively easy to implement.

Module 4: Identifying Transitions

This module deals with identifying good transition points in order to create new motion from existing data. The difficulties of this module lies in defining a "good" transition by using an evaluation function to determine how similar two poses between whom transition has to be created, are. This module is yet to be implemented.

Requirements

- **Define an evaluation function to determine how similar two poses are**, i.e., a distance to measure the difference between two poses. Because a character's pose is described by all of his/her joint angles and the position and orientation of his/her root node, a simple way to compare two poses is to calculate the sum of the distances between each joint angle and the position and orientation of the root node. Of course, this would not be a good pose metric since the weighting of each joint is the same whereas some joints are probably more important than others in choosing a "good" transition in the motion.

  Velocity or other information related to the continuity of a motion could be incorporated into the metric. The frames before and after the transition could also be taken into account when defining the metric. We need to experiment with several different options and find a metric that gives a good measurement of the perceptible difference of the motion in the two frames.

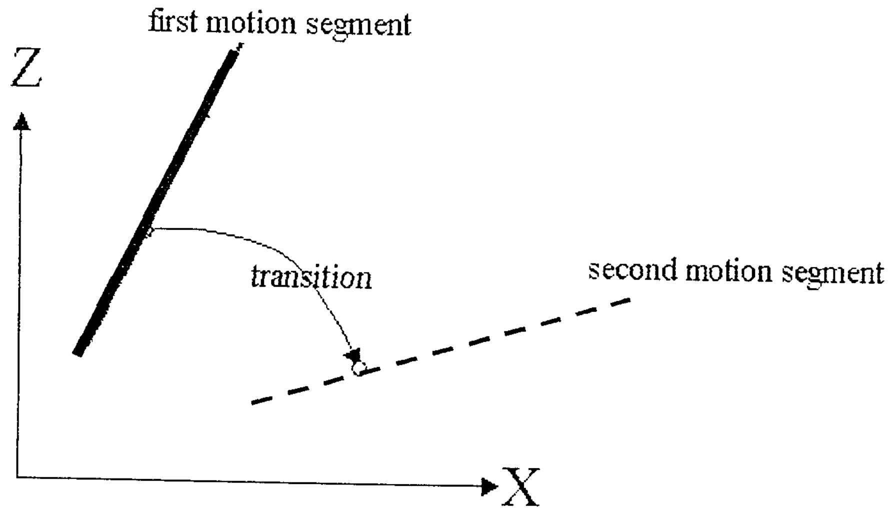

Fig. 9. Determining good transitions

In Fig. 9., solid and dotted lines represent the trajectory of the root node projected into XZ plane. Because we give a zero weight to the X and Z components of the root position and to the yaw orientation of the root, the transition shown in the figure will be declared "good" if all other joint angles are similar.

- **Design a fading algorithm to make the transition look smooth.** Although the distance function helps to find a close pose, the difference between two poses will usually be noticeable and will cause the transition look jerky. Blending the ending frames of the first motion and the starting frames of the second motion can reduce the jerkiness. Another alternative is to generate several additional transitional frames by interpolating between the ending frames of the segment before the transition and the starting frames of the segment after the transition.

- **Construct a directed graph to represent the transition.** Using the evaluation function, we can calculate the distance between pair of nodes (poses) and determine if an edge should be created between those two nodes. Because each motion capture file contains many frames, it would be quite time consuming to calculate the distance between for each pair of frames. We might have to down sample the data (only allow transitions only every few frames). We may also have to cull transitions quickly, if a major joint such as the hip is not close, then perhaps it is not worth checking the rest of the joints for similarity. We will also probably have to ignore links that would cause the motion to jump just a few frames ahead or behind of a current frame. We may have to prune transitions to select only local maxima or set all probabilities below some threshold to 0.

CHAPTER 6

Future Enhancements

CHAPTER 6
FUTURE ENHANCEMENTS

Three modules of the four have been completed. The fourth module is yet to be implemented. Also, various methods of creating "good" transitions among data could be discussed further.

One possible enhancement could be the following. Set a destination for the character, and plan a shortest path to reach the destination with smooth motion transitions. We can use a weighted edge to record the distance between two nodes. When traversing from one node to another, the accumulated cost on the path is a measurement of how much effort is required to generate a smooth transition as well as the distance traveled by the character between nodes. We can use a breadth first search to find a minimum cost path given these two error metrics.

Conclusion

CONCLUSION

Thus in conclusion, the implementation of motion graphs has proven to be largely successful. Three out of the four modules have been completed successfully. All aspects of the three modules have been incorporated fully and the bugs have been removed.

The project was a very challenging one to work with. This project presented a lot of hidden difficulties in terms of logic as well as implementation. This project proved to be mentally stimulating and intellectually satisfying. It gave a proper insight into the difficulties in the field of Computer Graphics and Animation. It also threw light on the most recent research currently being pursued in the same field.

On the whole, the project was very exciting and proved to be truly rewarding working on it.

References

REFERENCES

1. Gleicher M, *Retargetting motion to new characters*, In Proceedings of ACM SIGGRAPH '98.
2. Gould D, *Complete Maya Programming: An Extensive Guide to MEL and the C++ API,*
3. Hearn D, Baker P, *Computer Graphics,* Prentice Hall of India, Second Edition.
4. Kovar L, Gleicher M, Pighin F, *Motion Graphs*; In the Proceedings of ACM SIGGRAPH '02.
5. Kundert-Gibbs J, Lee P, *Mastering Maya,* BPB publications, Edition 2001.
6. Schildt H, *C++, A Beginner's Guide,* Tata McGraw-Hill, Edition 2002.
7. Schodl A, Szeliski R, Salesin D H, Essa I, *Video Textures.* In Proceedings of ACM SIGGRAPH 2000.
8. Stroustrup B, *The C++ Programming Language,* Pearson Education, Third Edition.
9. Wilkins R M, Kazmier C, *MEL Scripting for Maya Animators*
10. Xiang Z, Plastock R, *Computer Graphics,* McGraw-Hill, Second Edition.
11. www.aliaswavefront.com on 10th December 2003
12. www.mocap.cs.cmu.edu on 22nd December 2003

Appendix

SAMPLE CODE

MEL SCRIPT

```
//This MEL script is used to create the UI for the project
//-------------------------------------------------------
//-------------------------------------------------------
//This procedure is used to create the UI
 global proc loadUI(){

 string $my_window=`window -title "INITIAL CONSTRAINTS" -widthHeight 500 350`;
global string $fileInitial;
columnLayout;
   frameLayout -collapsable true -label "Initial Motion" -width 500;
    columnLayout;
         $fileInitial=`textFieldGrp -label "File name"`;
         button -label "Choose a file " -command "setFileName($fileInitial)";
         button -label "Save motion" -command "callInitial($fileInitial)" -align "right";
    setParent..;
  setParent..;

showWindow $my_window;

}

//-------------------------------------------------------
//This procedure calls the .mll file that creates an .AMC file
//of the initial motion that the user confirms
//
 global proc callInitial(string $fileInitial) {
$isloaded=`pluginInfo -query -loaded "optUIInitial.mll"`;
 if($isloaded==0){
  print("optUIInitial is not loaded in initial\n");
  loadPlugin optUIInitial.mll;
 }
  string $file=`textFieldGrp -q -fileName $fileInitial`;
  optUIInitial -c 1 -f $file; }


//-------------------------------------------------------
//This procedure sets the file name chosen by the user
//-------------------------------------------------------
global proc setFileName(string $fileInit)
{
  string $filename=`fileDialog`;
  textFieldGrp -e -text $filename $fileInit;
}

//-------------------------------------------------------
//End of MEL script file
//-------------------------------------------------------
```

CODE FOR CREATING AMC FILE

```
#include <iostream.h>
#include <math.h>
#include <stdio.h>
#include <fstream.h>
#include <maya/MFnPlugin.h>
#include <maya/MFnNurbsCurve.h>
#include <maya/MPointArray.h>
#include <maya/MDoubleArray.h>
#include <maya/MPoint.h>
#include <maya/MSelectionList.h>
#include <maya/MItSelectionList.h>
#include <maya/MSelectionList.h>
#include <maya/MItCurveCV.h>
#include <maya/MGlobal.h>
#include <maya/MDagPath.h>
#include <maya/MString.h>
#include <maya/MPxCommand.h>
#include <maya/MArgList.h>
#include <maya/MFileIO.h>
#include <maya/MItDag.h>
#include <maya/MMatrix.h>
#include <maya/MFnTransform.h>
#include <maya/MQuaternion.h>
#include <maya/MVector.h>
#include <maya/MObject.h>
#include <maya/MFnAttribute.h>
#include <maya/MAnimControl.h>
#include <maya/MFnAnimCurve.h>
#include <maya/MTime.h>
#include <maya/MItKeyFrame.h>

#ifndef min
static inline double
min (double a, double b)
{
        return (a < b ? a : b);
}
#endif

#ifndef max
static inline double
max (double a, double b)
{
        return (a > b ? a : b);
}
#endif

#define PI 3.14159265358979323846
enum NODES {root=0,lhipjoint,lfemur,ltibia,lfoot,ltoes,ltoes_end,
            rhipjoint,rfemur,rtibia,rfoot,rtoes,rtoes_end,
                                  lowerback,upperback,thorax,lowerneck,upperneck,head,head_end,

lclavicle,lhumerus,lradius,lwrist,lhand,lfingers,lfingers_end,

lthumb,lthumb_end,
rclavicle,rhumerus,rradius,rwrist,rhand,rfingers,rfingers_end,

rthumb,rthumb_end
};

MString NODES_NAME[]={"root","lhipjoint","lfemur","ltibia","lfoot","ltoes","ltoes_end",
            "rhipjoint","rfemur","rtibia","rfoot","rtoes","rtoes_end",

"lowerback","upperback","thorax","lowerneck","upperneck","head","head_end",

"lclavicle","lhumerus","lradius","lwrist","lhand","lfingers","lfingers_end",

"lthumb","lthumb_end",

"rclavicle","rhumerus","rradius","rwrist","rhand","rfingers","rfingers_end",

"rthumb","rthumb_end"
};
class optUIInitial : public MPxCommand
{

public:
        MFnDagNode  dagNode1;


                                optUIInitial();
        virtual         ~optUIInitial();

        MStatus         doIt( const MArgList& );
        MStatus         redoIt();
        MStatus         undoIt();
        bool            isUndoable() const;
        static          void* creator();


  double                toRad(double); //converts degrees(input) into radians (output)
        double          toDeg(double); //converts radians(input) into degrees (output)
  MStatus   findObject(MDagPath& , MString ); //returns the DagPath of the object when its name is the
input
        MStatus   nodeInfo(fstream );
        MStatus   initialPrint();
  char          filename[200];


private:
        fstream filep;
        MDagPath        fDagPath;

        MObject         fComponent;
        MPointArray     fCVs;
        double radius,pitch;

        int findJointNo(MString);
};

MStatus optUIInitial::doIt( const MArgList& args )
{
        MStatus status;
    MStatus stat;
        int code;
        double startFrame=1.0; //default start frame
        double endFrame=1.0;  //default end frame
        unsigned i;

        //scanning the arguments

        for ( i = 0; i < args.length(); i++ )
                if ( MString( "-c" ) == args.asString( i, &stat )
                        && MS::kSuccess == stat)
                {
                        int tmp=args.asInt(++i,&stat);
                        code=tmp;
                }
                else if ( MString( "-s" ) == args.asString( i, &stat )
                                && MS::kSuccess == stat)
                {
                        double tmp = args.asDouble( ++i, &stat );
                        if ( MS::kSuccess == stat )
                                startFrame = tmp;
                }
                else if ( MString( "-e" ) == args.asString( i, &stat )
                                && MS::kSuccess == stat)
                {
                        double tmp = args.asDouble( ++i, &stat );
                        if ( MS::kSuccess == stat )
                                endFrame = tmp;
                }
                else if ( MString( "-f" ) == args.asString( i, &stat )
                        && MS::kSuccess == stat)
                {
                        //setting the received file name
                        MString tmp=args.asString(++i,&stat);
                        strcpy(filename, tmp.asChar());
                }


                switch(code)
                {
                case 1:
                        //"Save Motion" button is clicked
                        initialPrint();
                        break;

                default:
                        cout<<"Something is wrong"<<endl;
                        return MS::kFailure;
                        break;
                }

        return MS::kSuccess;
}


//---------------------------------------------------------------------------------------------------
//This section prints all info about the joints for all the frames corresponding to the initial motion
```

```
//----------------------------------------------------------------------------------------------------------------
MStatus optUIInitial::initialPrint()
{

            fstream tempFile;
            tempFile.open(filename, ios::out);
            if(!tempFile)
            {
                        cout<<"Error opening file in initialPrint"<<endl;
            }
   else
            {
                        tempFile<<"#!OML:ASF
F:\\VICON\\USERDATA\\INSTALL\\rory3\\rory3.ASF\n:FULLY-SPECIFIED\n:DEGREES\n";
            }


            MAnimControl animctrl;
            MTime startFrame=animctrl.minTime();
            MTime endFrame=animctrl.maxTime();

            double inttime;

   for(MTime time=startFrame;time<=endFrame;time++)
            {

                        MGlobal::viewFrame(time);
                        inttime=time.value();
                        tempFile<<inttime;
                        nodeInfo(tempFile); // this function prints info of all the joints into the file stream
specified
            }

   MGlobal::viewFrame(startFrame);
            tempFile.close();
            return MS::kSuccess;
}


//----------------------------------------------------------------------------------------------------------
//----------------------------------------------------------------------------------------------------------
//----------------This section traverses the hypergraph and prints info of all the nodes ---------------------
//----------------------------------------------------------------------------------------------------------

MStatus optUIInitial::nodeInfo(fstream tempFile)
{
            MDagPath dagpath;
            MStatus status;
            MString name;
            MObject obj;
            MObject rxAttr;
            MObject ryAttr;
            MObject rzAttr;

            const char* charname;


            double rotationrad[3];
```

```
        double rotationdeg[3];

        MVector translation;

        bool lockedx;
        bool lockedy;
        bool lockedz;

        for(int i=0;i<37;i++)
        {
                status=findObject(dagpath,NODES_NAME[i]);
                MFnDagNode dagNode(dagpath);
                name=dagNode.name();
                charname=name.asChar();
                obj=dagpath.node();
                MFnDependencyNode fnDependNode(obj);

                MFnAnimCurve animcurve(obj);
                rxAttr=fnDependNode.attribute(MString("rotateX"),&status);
                ryAttr=fnDependNode.attribute(MString("rotateY"),&status);
                rzAttr=fnDependNode.attribute(MString("rotateZ"),&status);

                MFnAttribute fnattribute(obj);
        MFnTransform fntransform(dagpath);

                MPlug rxPlug(obj,rxAttr);
                MPlug ryPlug(obj,ryAttr);
                MPlug rzPlug(obj,rzAttr);

                lockedx=rxPlug.isLocked();
                lockedy=ryPlug.isLocked();
                lockedz=rzPlug.isLocked();


                MTransformationMatrix::RotationOrder order=MTransformationMatrix::kXYZ;
                fntransform.getRotation(rotationrad,order,MSpace::kTransform);

                for(int j=0;j<3;j++)
                        rotationdeg[j]=toDeg(rotationrad[j]);

    if(!(lockedx&&lockedy&&lockedz)){
                        tempFile<<"\n"<<charname<<"\t";
                        if(i==0){
                        translation=fntransform.translation(MSpace::kTransform);
        tempFile<<translation.x*(0.45/2.54)<<"\t"<<translation.y*(0.45/2.54)<<"\t"<<translation.z*(0.45/
2.54)<<"\t";
                        }

                        if(!lockedx)
                        {
                                tempFile<<rotationdeg[0]<<"\t";
                        }
                        if(!lockedy)
                        {
                                tempFile<<rotationdeg[1]<<"\t";
                        }
                        if(!lockedz)
                        {
                                tempFile<<rotationdeg[2]<<"\t";
                        }

                }

        }
        tempFile<<"\n";
        return MS::kSuccess;
}


MStatus optUIInitial::findObject(MDagPath& objectPath, MString objectName)
{
        const MItDag::TraversalType traversalType = MItDag::kDepthFirst;
        MFn::Type filter = MFn::kJoint;
        MStatus  status;

        //Scan entire DAG to root node for character and for human in normal standing position
        //human nodes should be under group - HUMAN_STAND_GROUP_NAME
        //character nodes should be under group - CHAR_GROUP_NAME
        //
        //MItDag dagIterator( traversalType, filter, &status);
        MItDag dagIterator;

        for ( ; !dagIterator.isDone(); dagIterator.next() )
        {
                MDagPath dagPath;

                status = dagIterator.getPath(dagPath);
                if ( !status ) {
                        status.perror("MItDag::getPath");
                        return status;
                }

                MFnDagNode dagNode(dagPath, &status);
                if ( !status ) {
                        status.perror("MFnDagNode constructor");
                        return status;
                }

                if (dagNode.name() == objectName)
                {
                        objectPath = dagPath;
                        return MS::kSuccess;
                }
        }
        return MS::kFailure;

}


MStatus optUIInitial::redoIt()
{
        unsigned                        i, numCVs;
        MStatus                    status;
        MFnNurbsCurve   curveFn( fDagPath );

        numCVs = curveFn.numCVs();
        status = curveFn.getCVs( fCVs );
        if ( MS::kSuccess != status )
        {
                cerr << "redoIt: could not get cvs status: " << status << endl;
                return MS::kFailure;
        }

        MPointArray                points(fCVs);
        for (i = 0; i < numCVs; i++)
                points[i] = MPoint( radius * cos( (double)i ),
                                                                  pitch * (double)i,
                                                                  radius * sin( (double)i ) );
        status = curveFn.setCVs( points );
        if ( MS::kSuccess != status )
        {
                cerr << "redoIt: could not setCVs status: " << status << endl;
                fCVs.clear();
                return status;
        }

        status == curveFn.updateCurve();
        if ( MS::kSuccess != status )
        {
                cerr << "redoIt: updateCurve() failed status: " << status << endl;
                return status;
        }

        return MS::kSuccess;
}

MStatus optUIInitial::undoIt()
{
        MStatus                 status;

        MFnNurbsCurve curveFn( fDagPath );
        status = curveFn.setCVs( fCVs );
        if ( MS::kSuccess != status)
        {
                cerr << "undoIt: array length is " << fCVs.length()
                    << "bad status: " << status << endl;
                return status;
        }

        status = curveFn.updateCurve();
        if ( MS::kSuccess != status )
        {
                cerr << "undoIt: updateCurve() failed status: " << status << endl;
                return status;
        }

        fCVs.clear();
        return MS::kSuccess;
}
```

```
void* optUIInitial::creator()
{
	return new optUIInitial();
}

optUIInitial::optUIInitial()
{

}

optUIInitial::~optUIInitial()
{
	fCVs.clear();
	// Note that we do nothing with fComponent which is owned by Maya.
}

bool optUIInitial::isUndoable() const
{
	return true;
}

MStatus initializePlugin( MObject obj )
{
	MStatus   status;
	MFnPlugin plugin( obj, "Alias|Wavefront - Example", "3.0", "Any");

	status = plugin.registerCommand( "optUIInitial", optUIInitial::creator );
	if (!status) {
		status.perror("registerCommand");
		return status;
	}

	return status;
}

MStatus uninitializePlugin( MObject obj )
{
	MStatus   status;
	MFnPlugin plugin( obj );

	status = plugin.deregisterCommand( "optUIInitial" );
	if (!status) {
		status.perror("deregisterCommand");
		return status;
	}

	return status;
}

double optUIInitial::toRad(double degrees)
{
	return degrees*PI/180.0;
}


double optUIInitial::toDeg(double radians)
{
	return radians*180.0/PI;
```

```
}

//This function takes the name of the joint and returns the joint no corresponding to the name
int optUIInitial::findJointNo(MString name)
{
 if(name=="root")
          return(0);
 if(name=="lhipjoint")
          return(1);
 if(name=="lfemur")
          return(2);
 if(name=="ltibia")
          return(3);
 if(name=="lfoot")
          return(4);
 if(name=="ltoes")
    return(5);
 if(name=="ltoes_end")
          return(6);
 if(name=="rhipjoint")
          return(7);
 if(name=="rfemur")
          return(8);
 if(name=="rtibia")
          return(9);
 if(name=="rfoot")
          return(10);
 if(name=="rtoes")
          return(11);
 if(name=="rtoes_end")
          return(12);
 if(name=="lowerback")
          return(13);
 if(name=="upperback")
    return(14);
 if(name=="thorax")
          return(15);
 if(name=="lowerneck")
          return(16);
 if(name=="upperneck")
          return(17);
 if(name=="head")
          return(18);
 if(name=="head_end")
          return(19);
 if(name=="lclavicle")
          return(20);
 if(name=="lhumerus")
          return(21);
 if(name=="lradius")
          return(22);
 if(name=="lwrist")
          return(23);
 if(name=="lhand")
          return(24);
 if(name=="lfingers")
    return(25);
```

```
if(name=="lfingers_end")
        return(26);
if(name=="lthumb")
        return(27);
if(name=="lthumb_end")
        return(28);
if(name=="rclavicle")
        return(29);
if(name=="rhumerus")
        return(30);
if(name=="rradius")
        return(31);
if(name=="rwrist")
        return(32);
if(name=="rhand")
        return(33);
if(name=="rfingers")
   return(34);
if(name=="rfingers_end")
   return(35);
if(name=="rthumb")
        return(36);
if(name=="rthumb_end")
        return(37);
return(-1);
```

RESULTS

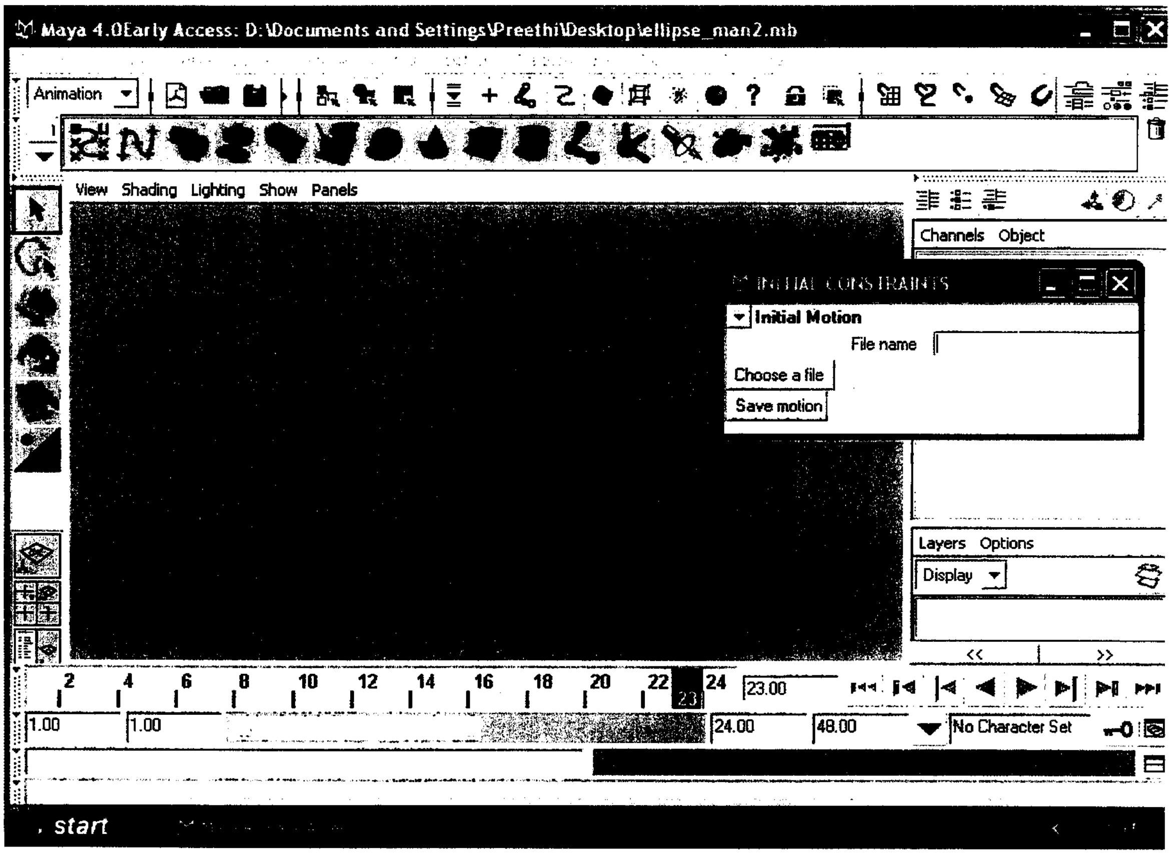

Fig 10. Initial Interface for Creation of AMC file

The above figure (Fig 10) is the interface created using MEL script for Module 1. This interface is invoked by clicking the "mel" button on the toolbar. This interface is used to convert the animation created using the animation software Maya into an AMC file. The "INITIAL CONSTRAINTS" box was created using MEL scripting language. Using this a file can be chosen or if a file name is entered, then a new file is created which is the AMC file. Clicking on "Save motion" button saves the animation into the specified AMC file. All other buttons and resources seen in the picture is a part of Maya.

AMC FILE

A part of the AMC file created is shown below. Only details up to 5 frames are given below. The data used in this project (motion capture data) was obtained from mocap.cs.cmu.edu.

The database was created with funding from NSF EIA-0196217.

```
#!OML:ASF F:\VICON\USERDATA\INSTALL\rory3\rory3.ASF
:FULLY-SPECIFIED
:DEGREES
1
root -2.88351 17.1557 15.9323 -154.737 -18.9506 -179.665
lowerback -10.1321 2.37497 -0.0246197
upperback -8.11655 2.34364 -0.192357
thorax -2.77034 0.813939 0.00705613
lowerneck -13.1253 2.30464 -0.545957
upperneck 42.2538 -1.49169 0.658922
head 17.9267 -2.11971 -0.358962
rclavicle 9.31803e-016 9.14409e-015
rhumerus -44.2164 -23.9018 -59.8092
rradius -2.40984
rwrist 31.0716
rhand -14.0311 5.43764
rfingers 7.12502
rthumb 12.0992 -24.1925
lclavicle 9.31803e-016 9.14409e-015
lhumerus -50.2403 23.9934 54.4774
lradius 22.4541
lwrist -6.32816
lhand -18.8932 21.488
lfingers 7.12502
lthumb 7.40824 51.3383
rfemur -2.25394 1.02385 23.8502
rtibia 2.22639e-014
rfoot -25.1731 -30.8075
rtoes 13.2871
lfemur -45.1631 5.03063 -30.8943
ltibia 30.2653
lfoot -29.462 23.9888
ltoes 18.5768
2
root -2.91308 17.16 15.8956 -154.486 -19.1177 -179.804
lowerback -10.214 2.19692 -0.22376
upperback -8.2164 2.10221 -0.145691
thorax -2.84149 0.694638 0.149842
lowerneck -13.2537 2.4938 -0.687804
upperneck 42.3973 -1.21831 0.764256
head 18.0442 -2.02703 -0.240486
rclavicle -1.60146e-014 -1.41137e-014
rhumerus -44.4103 -23.578 -59.9007
rradius -2.33539
rwrist 31.4043
```

```
rhand -13.6877 5.37642
rfingers 7.12502
rthumb 12.4302 -24.234
lclavicle -1.60146e-014 -1.41137e-014
lhumerus -50.2049 23.416 54.9655
lradius 22.5953
lwrist -6.26372
lhand -18.8617 21.766
lfingers 7.12502
lthumb 7.43863 51.6152
rfemur -2.24077 0.84122 23.9044
rtibia 6.36111e-015
rfoot -24.6502 -31.4862
rtoes 12.5869
lfemur -45.1272 4.80614 -31.0145
ltibia 30.1751
lfoot -29.5262 23.827
ltoes 18.7036
3
root -2.9609 17.1591 15.8451 -154.607 -19.1033 -179.644
lowerback -10.2653 1.97803 -0.124084
upperback -8.05465 1.82523 0.120545
thorax -2.64637 0.566645 0.351478
lowerneck -13.8557 2.75371 -0.950478
upperneck 42.6228 -0.667803 0.675922
head 18.3339 -1.85891 -0.140303
rclavicle 1.69837e-014 1.80894e-014
rhumerus -44.0309 -22.916 -60.157
rradius -2.20261
rwrist 31.5258
rhand -13.2925 5.38749
rfingers 7.12502
rthumb 12.8112 -24.1994
lclavicle 1.69837e-014 1.80894e-014
lhumerus -50.4256 24.239 54.7417
lradius 22.4642
lwrist -6.99162
lhand -18.9588 21.2478
lfingers 7.12502
lthumb 7.34493 51.1003
rfemur -1.81893 0.925172 24.1362
rtibia -1.27222e-014
rfoot -23.658 -32.2498
rtoes 11.1169
lfemur -44.9732 4.66093 -30.8287
ltibia 30.5859
lfoot -30.1647 23.7362
ltoes 18.3219
4
root -3.02942 17.1533 15.7703 -155.765 -18.6991 -178.909
lowerback -10.1245 2.7374 0.935208
upperback -7.90655 2.78754 -0.35936
thorax -2.49498 1.00681 -0.664172
lowerneck -12.5051 2.35001 -0.259361
upperneck 42.3814 -1.75043 0.900771
head 17.8447 -2.18405 -0.351316
rclavicle -3.37002e-014 -1.0635e-014
rhumerus -45.827 -23.9608 -59.4646
```

```
rradius -1.91983
rwrist 31.3542
rhand -12.9044 5.21169
rfingers 7.12502
rthumb 13.1853 -24.3514
lclavicle -3.37002e-014 -1.0635e-014
lhumerus -49.7016 22.9215 54.7195
lradius 22.0732
lwrist -8.1015
lhand -18.6642 19.7856
lfingers 7.12502
lthumb 7.6293 49.628
rfemur -0.337928 1.52344 24.759
rtibia 4.13472e-014
rfoot -22.7604 -32.9678
rtoes 9.50333
lfemur -43.7968 5.22121 -30.1649
ltibia 30.9383
lfoot -30.82 23.6336
ltoes 17.7088
5
root -3.06741 17.1502 15.7114 -156.2 -18.4297 -178.831
lowerback -10.1389 2.44148 0.595923
upperback -7.39105 2.48785 0.103782
thorax -1.98609 0.915045 -0.0120953
lowerneck -13.2045 2.50109 -0.706906
upperneck 42.3905 -1.20326 0.7252
head 18.026 -2.03203 -0.252631
rclavicle -5.05162e-014 3.57812e-015
rhumerus -44.4617 -23.3281 -59.8925
rradius -2.53524
rwrist 31.0532
rhand -12.7184 5.3027
rfingers 7.12502
rthumb 13.3647 -24.2487
lclavicle -5.05162e-014 3.57812e-015
lhumerus -49.8351 24.0371 54.8596
lradius 22.2236
lwrist -8.59963
lhand -18.4642 19.1541
lfingers 7.12502
lthumb 7.82227 48.9894
rfemur 0.561245 1.77604 24.9213
rtibia 1.59028e-014
rfoot -21.8394 -33.5143
rtoes 9.13338
lfemur -43.1981 5.15657 -29.9628
ltibia 31.2881
lfoot -31.7494 23.4067
ltoes 18.34
```

ASF-AMC-VIEWER

The figure below (Fig 11) shows the viewer, i.e., the interface that is used to view the simple motion graph created by reading the AMC file. For further details regarding the functionalities supported by the viewer, refer to Module 3 in the Detailed Design section (Page No 46).

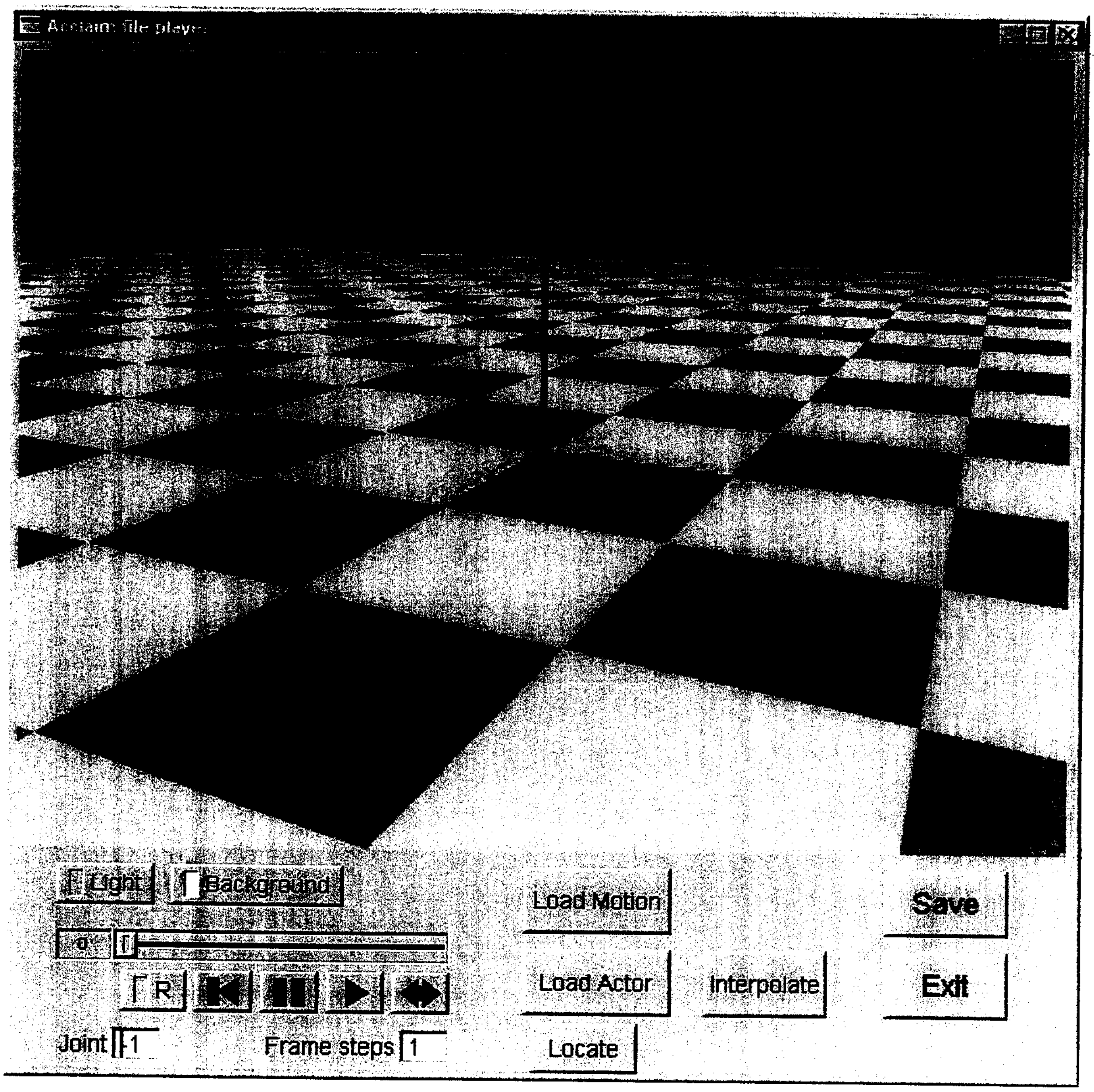

Fig 11. ASF – AMC - Viewer

The following figures (Fig 12 to Fig 19) show the results of the simple motion graph created. Only eight frames from a sequence of 1354 frames have been shown below.

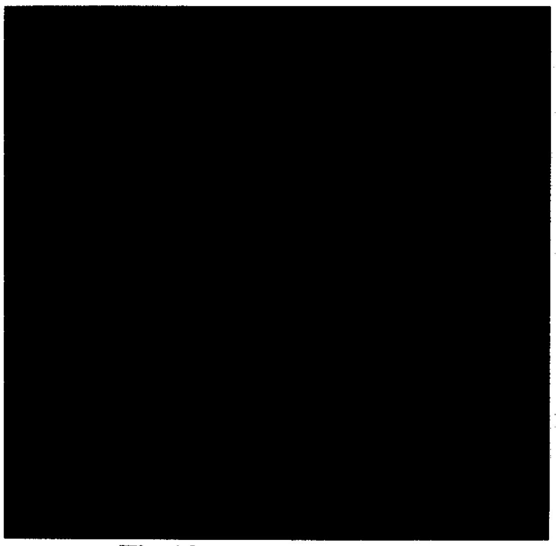
Fig 12 Frame 1087

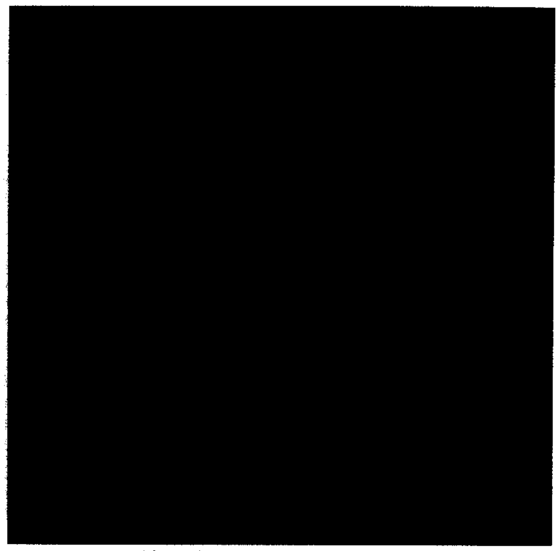
Fig 13 Frame 1126

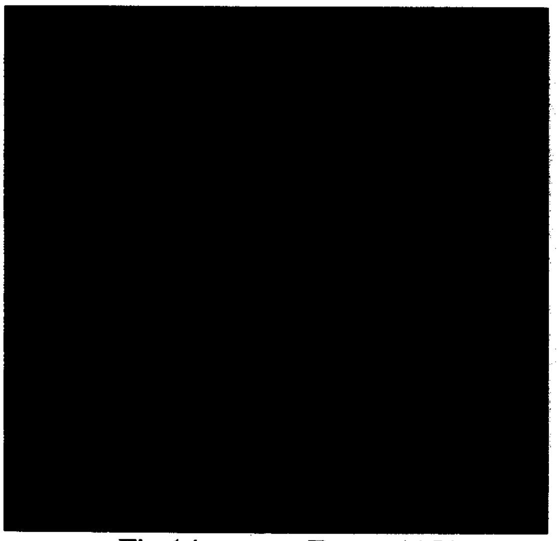
Fig 14 Frame 1151

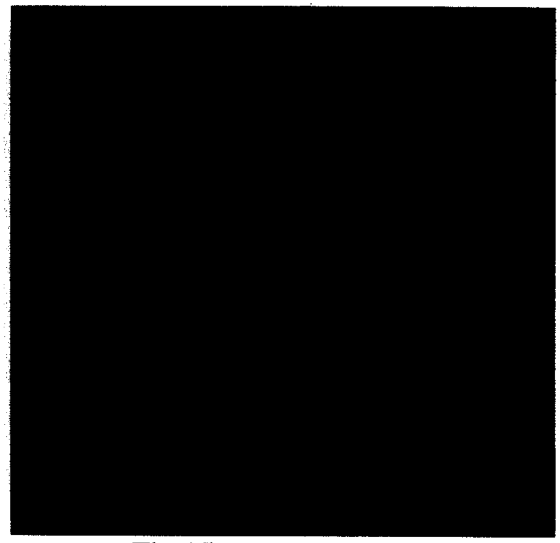
Fig 15 Frame 1220

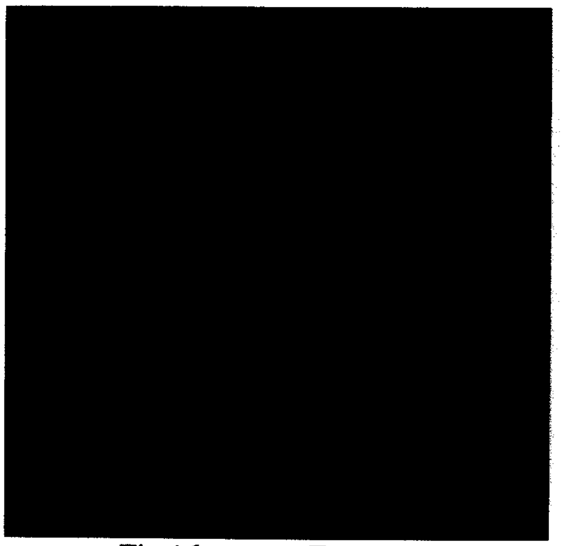
Fig 16 Frame 1291

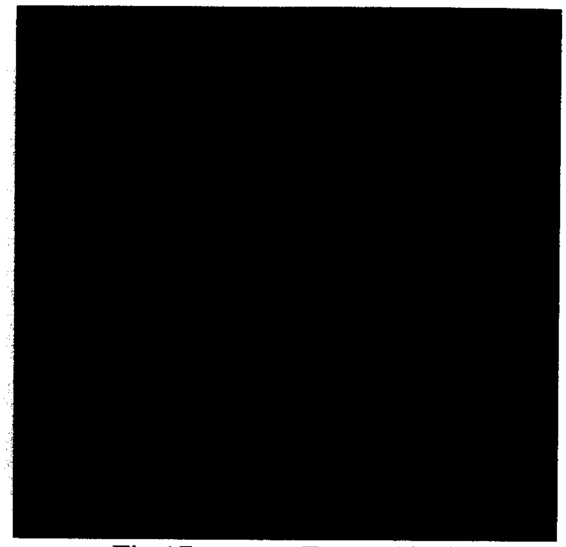
Fig 17 Frame 1316

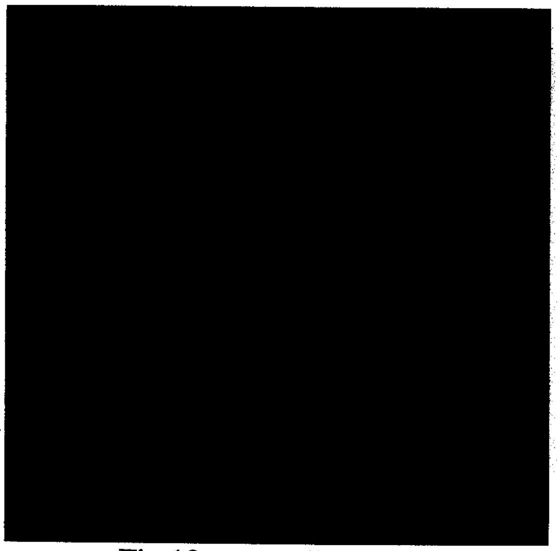
Fig 18 Frame 1336

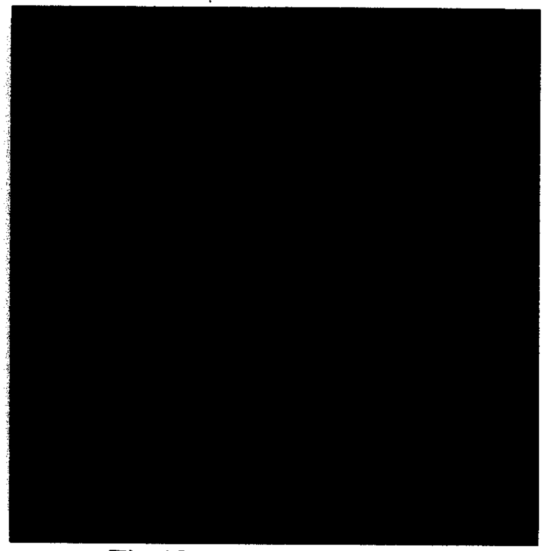
Fig 19 Frame 1347