R2Dtool

"Auto-Rigging Tool with emphasis on ease of initial set up, and incorporating a 2D limb placement"

Leigh M Evans (d1179790)

NCCA BACVA3

20007

Abstract

This study is an investigation into the creation of an automatic CG rig generator, using a simplistic setup. The method chose was the simplest possible; placing coloured dots on the front and side image of a character design. The image was processed with a program written in Python, and the information passed onto MAYAs programming language, MEL. The resulting product was multi-resolution rig with many features identified from researching current rigging techniques.

Summary

With an initial idea into rigging in 3D using a simplistic 2D setup, a system was designed where joints could be placed within a 3D environment to form a character rig, based solely on images of the character (front and side views). The coordinates were generated for all the major joints by placing coloured dots onto the image, which was then processed by a short Python script. The resulting text file was then read by a MEL script that placed the major joints into the scene, then inferred the position of all the other joints and controllers necessary for a fully functioning rig. The features of the rig were suggestive of current rigging trends, as shown by research. Despite a successful study into producing a working artifact, more development is needed to produce a truly distributable tool (and to iron out some existing problems), with a usable interface, and rigorous stress testing of the resulting rig has yet to be studied.

1. Introduction

1.1 What is a Rig?

> '…rigging involves many different skills. It is half art and half science'
> **"The Art of Rigging – Volume 1", Kiaran Ritchie et al – CG Toolkit , 2005 [1]**

Movement (or motion), whether it is on a cellular level or in propelling an entire organism, is generally regarded by biologists as being one of the defining factors of life; the sheer diversity of which dictates a massive array of adaptations and methods used in achieving locomotion. Likewise, in creating the conditions for motion artificially, we as humans can call upon the simplicity of the wheel to the elegance of the aerofoil.

It is in the complexities of sailing that we find the literal relationship between 'physical world' motion and computer graphics simulation.
A sailing 'rig' or 'rigging' (from the Anglo-Saxon *wrigan***,** meaning "to clothe") describes the interconnecting ropes, devices and structures used in harnessing the natural power of the wind inorder to convert it into a controllable kinetic energy. A 3D CG character rig does just this; it provides the means to 'capture' the creative energy of the animator, by a hierarchical system of joints and constraints.

1.2 The basic mechanics involved

> '…programming, albeit at a very high level…a [creature rig] is simply a large collection of

meticulously organized and connected nodes'

"The Art of Rigging – Volume 1", Kiaran Ritchie et al – CG Toolkit , 2005 [1]

As a general principle, adding a skeletal structure to a 3D CG character involves the strategic placement of 'joints', much like the joints found in a biological skeletal structure. AutoDesk's MAYA software uses a 'joint' node hierarchy system to describe 'skeleton chains'; the CG equivalent of a limb. [see fig 1.1]

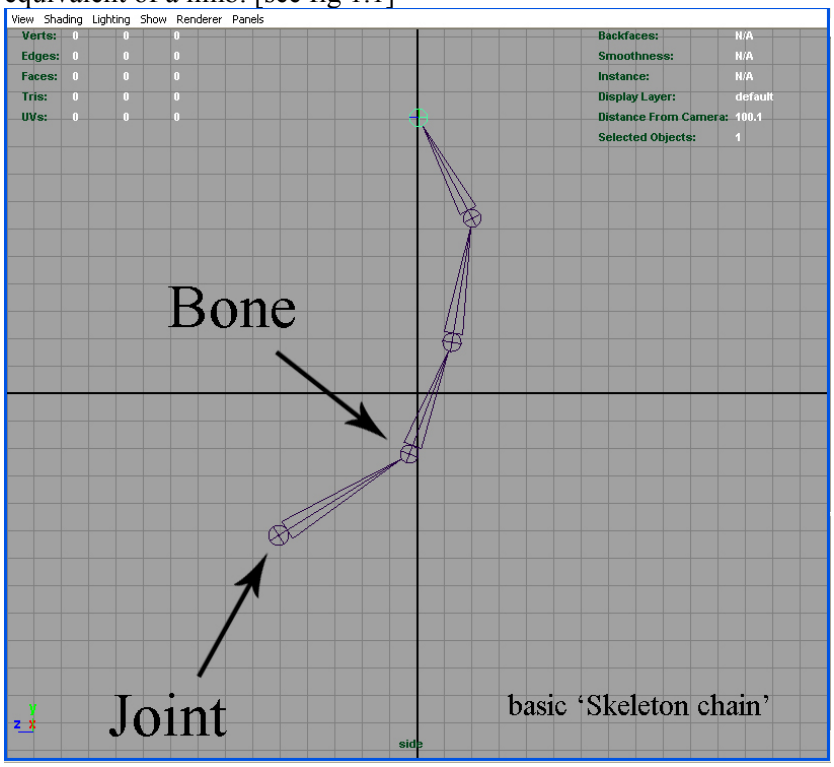

fig.1.1 Image to show a basic skeleton chain

Each node exhibits a 'parent-child' relationship with its immediate neighbour, establishing 'inheritance' within the skeleton chains, and creating the conditions for **Forward Kinematics (FK)**. This is best explained with the simple example of a leg:

- The foot moves into position, due to the rotation of the ankle
- The ankle moves due to the rotation of the knee
- The knee moves due to the rotation of the hip [see fig 1.2]

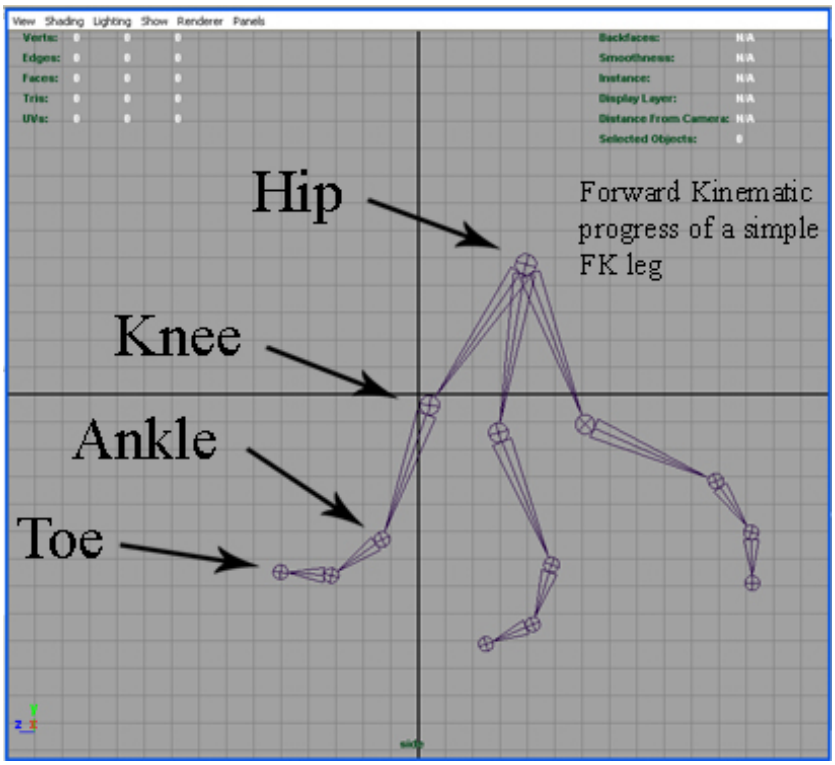

fig.1.2 Image to show Forward Kinematic movement using the example of a leg skeleton chain

Forward Kinematic behaviour alone, is not sufficient to adequately emulate the full range of movement available to a physical skeletal structure. When the last joint in a skeleton chain hierarchy needs to remain stationary but the rest of the body continues to move about it, **Inverse Kinematic (IK)** behaviour is required:

- The hands are placed on a block
- The body positions itself to push the block
- The hands remain stationary throughout the process, despite movement from parent joint nodes [see fig 1.3]

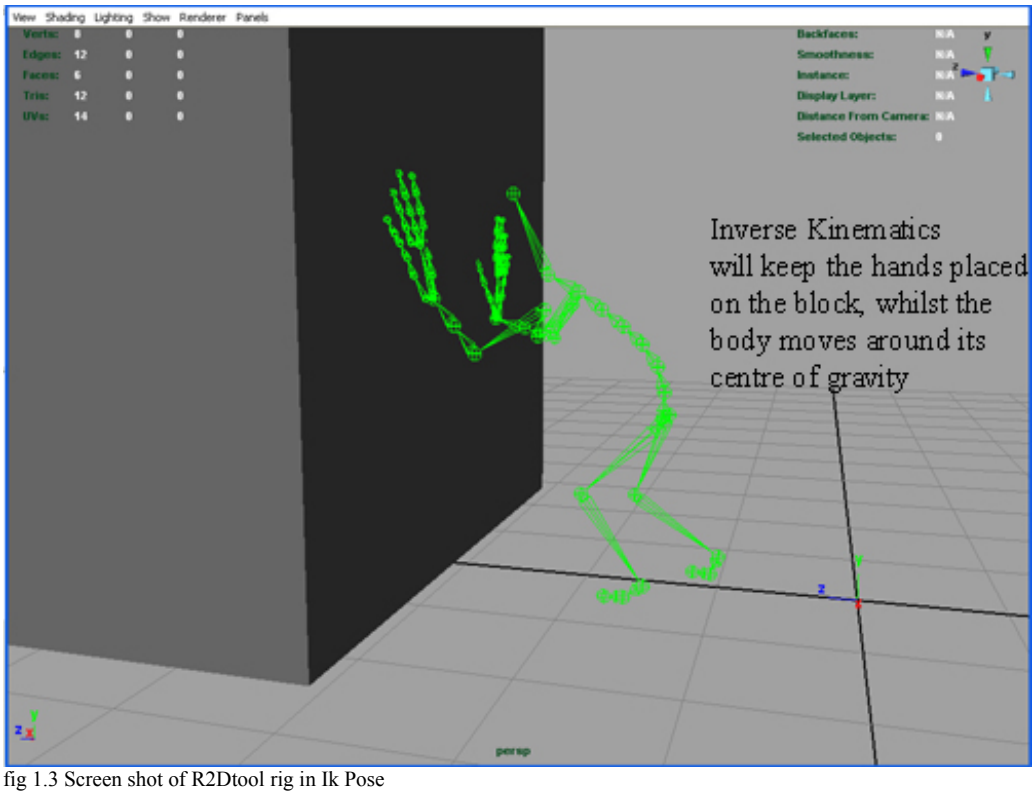

fig 1.3 Screen shot of R2Dtool rig in Ik Pose

MAYA employs several IK solver algorithms to calculate the resulting joint rotations of each joint node within a skeleton chain. In order to be used, Inverse Kinematics must be explicitly added to skeleton chains through the use of IK handles. [see fig 1.4]

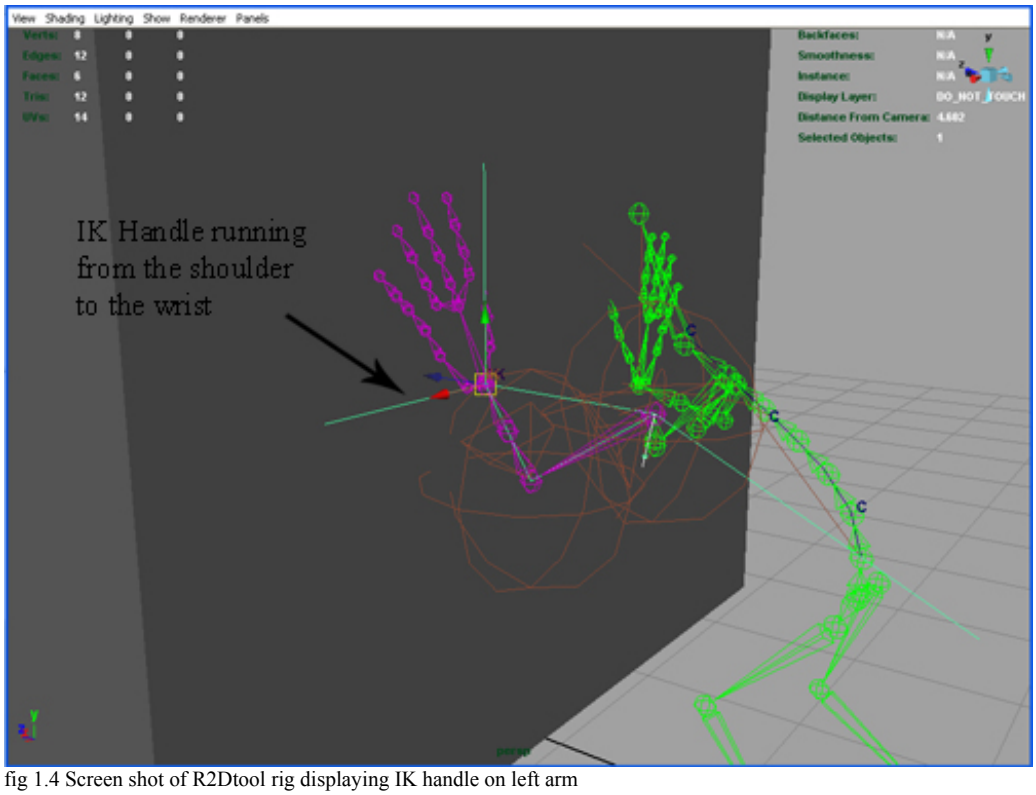

fig 1.4 Screen shot of R2Dtool rig displaying IK handle on left arm

1.3 A simple bipedal rig and required functionality

> 'In the same way that an athlete relies on good bone structure and toned musculature to maximise their performance, a well built rig lies behind any good animation'
> **"3D World", Kira-Anne Pelican , 2007 [2]**

In the creation of a basic bipedal rig (arguably the most common example), **Kiaran Ritchie et al (2005)** state that 'while every set-up artist seems to have their own joint placement conventions, everybody agrees on at least a basic layout for a biped creature'. The convention follows the basic joint positioning and subsequent skeleton chains for the following elements of the skeleton:

- Spine (3 to 5, or sometimes even 10 to 20 joints) with a COG joint (centre of gravity, about which the entire rig can move)
- Neck
- Head

R2Dtool

- Jaw
- Feet
- Hands
- Legs
- Arms

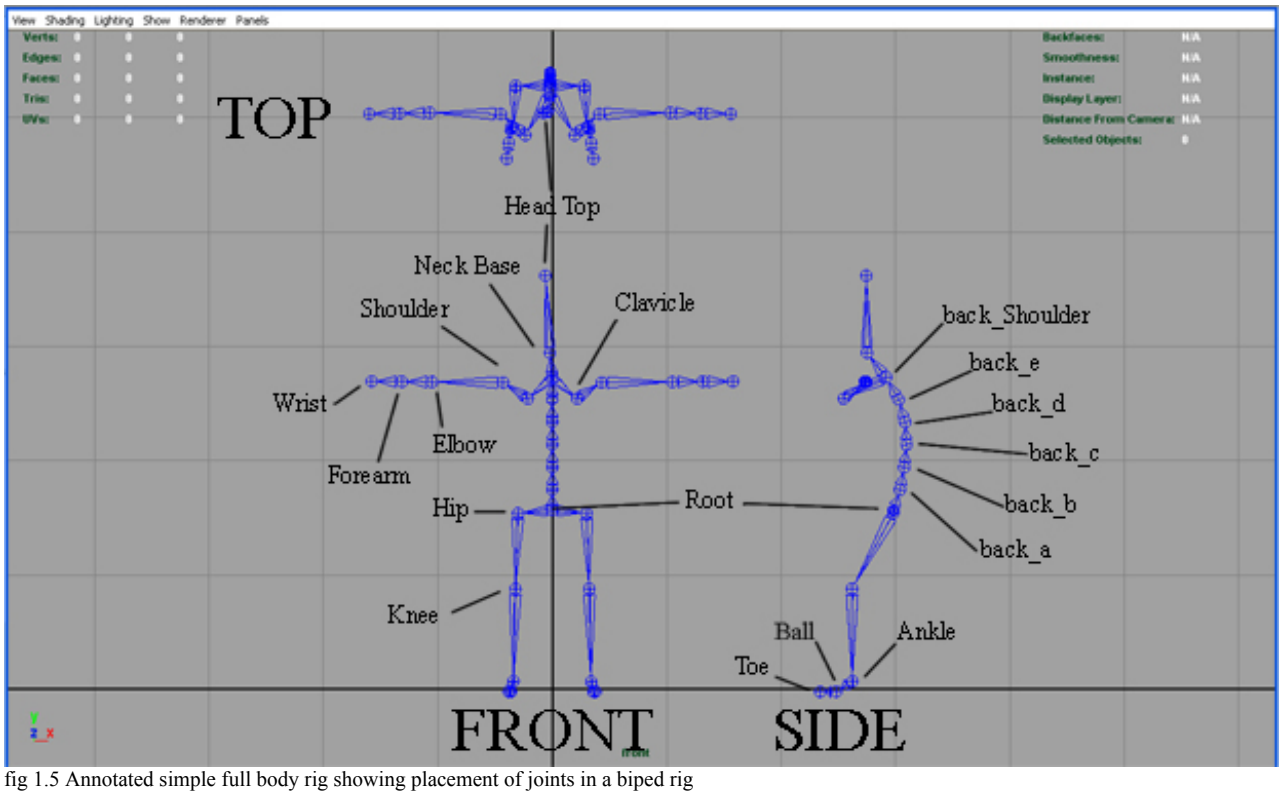

fig 1.5 Annotated simple full body rig showing placement of joints in a biped rig

1.3.1 An Industry perspective

CG character rigging resides within the fast paced and 'ever changing' field of Computer Graphics. In considering research or a departure for investigating a new and innovative alternative, it is essential to talk directly with those involved, in situ.

Karl Wickens is a senior artist at EA games. Below is a transcript from the correspondence received from Karl, when he was contacted and asked to comment on the subject of CG character rigging and the aims of this study.

R2Dtool

> *"…I believe most animation tools provide the functionality to blend mocap data over hand animation in Layers, so this might not be something you want to solve explicitly in the tool though provide support for mocap skeletons might be worth while. One important feature, that I am aware of, is the ability to switch between Inverse and Forward Kinematics while animating, to provide global and precise solutions for poses…"*

Phil Warner, lead artist at Codemasters Internal development studios, was also asked for his thoughts on the subject and the proposition of a rigging tool working only on images

> *"… I do think that this would be a useful tool. Realistically, trying to make a tool that totally eradicates the need for user involvement in the rigging is unlikely. A tool that spits out a correctly proportioned rig would be useful though if it can get you 80% of the way there.*
>
> *One thing I would say is that most companies, or even most games, will have their own rig specifications. As such, a useful feature would be to be able to feed user-defined rigs into the tool and have them adjust accordingly. Maybe it would work off naming conventions for example to recognise the various joints. This functionality would be far more useful than a tool that gives you a pre-determined rig that may not be suitable for a project's need*."

1.3.2 "Squash and Stretch"

> 'Of all the animation principles that have been ported into the digital age, squash and stretch is perhaps the most difficult to replicate inside a computer'
> **"The Art of Rigging – Volume 1", Kiaran Ritchie et al – CG Toolkit , 2005 [1]**

NCCA Animation tutor, Steve Harper, was asked for his views on a 2D rigging system, which could be applied in 3D. His suggestions on some of the desired elements included 'an intuitive user interface, making joint selection easier', and also the addition of 'Squash and Stretch' functionality; a method of animation whereby the character can distort during movement to display a 'cartoon' style movement, emphasising speed, or weight in an unrealistic yet stylistic fashion. [see fig 1.6]

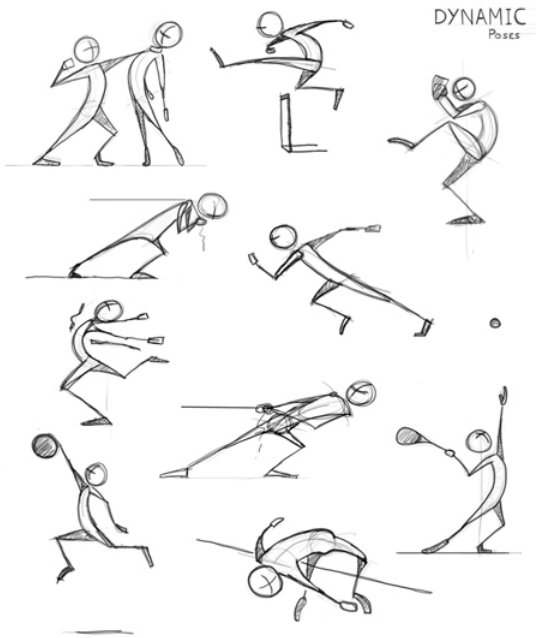

fig 1.6 Image depicting the principles of 'Squash and Stretch' in dynamic poses
http://www.keyframer.dreamhosters.com/blog/wp-content/uploads/2006/08/105_dynamicsketches.jpg

Vector based animation systems such as Anime Studio 5 (formerly called 'Moho') by Lost Marble, already effectively employ 2D joint hierarchy systems. A brief online video tutorial by Steve Ryan, on how to rig in Anime Studio 5 can be found at the following website.

http://www.steveryan.net/Tutorials/Moho/Moho_basic_bone_rigging.html

CG Toolkits "The Art of Rigging – Volume 1"[1] (pp38-54), includes a chapter on implementing 'Squash and Stretch' to limbs via a MEL script. Due to the complexity of this project, this functionality will not be applied, but is considered for further study.

1.3.3 "Body Movin" – a recent 3D World article

> 'The general approach to character rigging across most 3D animation software is similar'
>
> **"3D World", Kira-Anne Pelican , 2007 [2]**

A recent article by Kira-Anne Pelican, in '3D World' magazine[2], highlights the 3 approaches to rigging, in order of increasing complexity and functionality:

1. **'An all-encompassing, hierarchical set of joints'**

R2Dtool

"Auto-Rigging Tool with emphasis on ease of initial set up, and incorporating a 2D limb placement"

Basic, simple and easy to implement.

2. A Modular rigging strategy

A more complex set up, involving separate skeleton chains, constrained in some way to each other. A simple example of this is a Reverse Foot rig, used in conjunction with an IK system for greater control over foot deformation. [see fig 1.7]

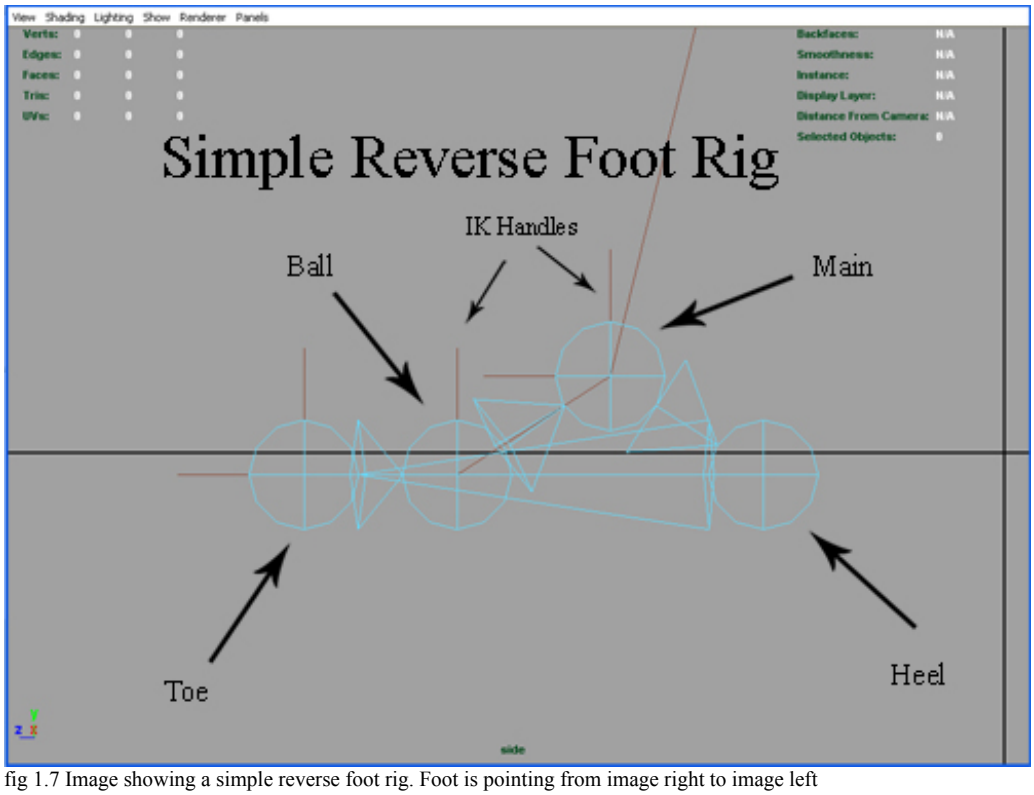

fig 1.7 Image showing a simple reverse foot rig. Foot is pointing from image right to image left

3. A Multi-Resolution rig

A versatile but potentially very complex set up utilising various duplicate rig sets for various limbs, and using them to drive a non-rigged skeleton, to which the character mesh is bound. [see fig 1.8]

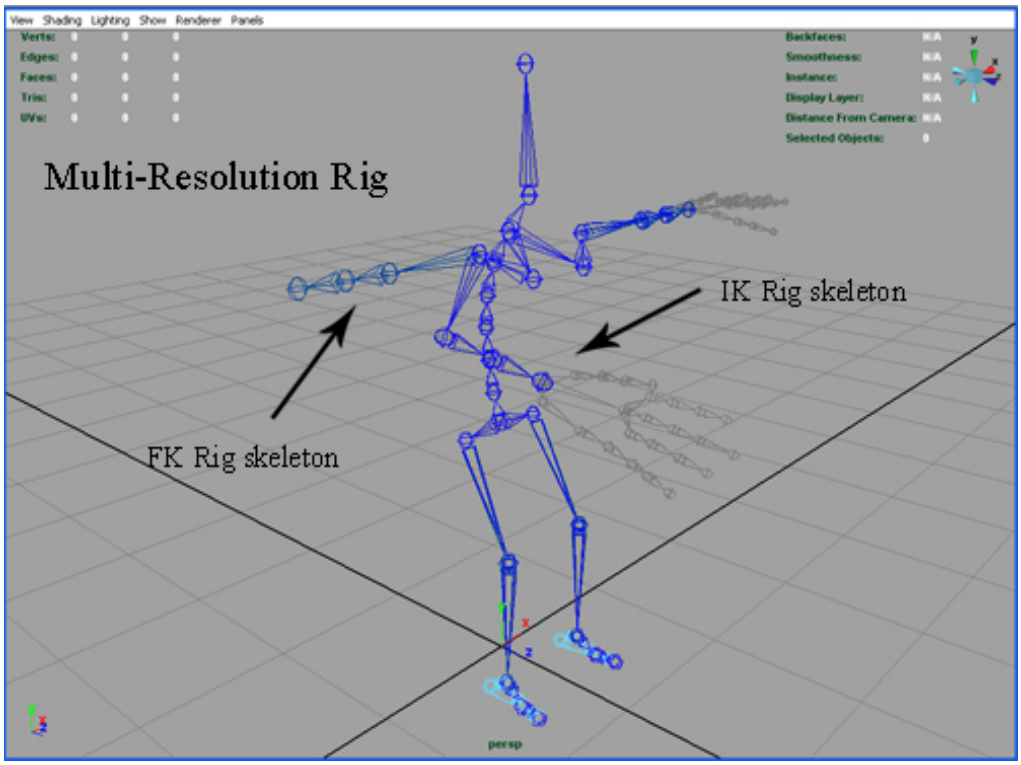

fig 1.8 Screenshot of R2Dtool multi resolution rig set. Here the IK and FK arm skeleton chains for the right arm show the two animation solutions available for the selected limb

The article also gave valuable insight into several general rigging principles and tips that can be applied across the various 3D software packages; the specific examples were given using Maya, SoftImage, XSI, 3DS Max, Lightwave 3D and MotionBuilder.

- **Aiding in Joint selection**
  The addition of NURBS curve 'controllers' can greatly improve the usability of a rig system The curves in MAYA are non-renderable, and are easy to change in colour, size and location**.** [see fig 1.9]

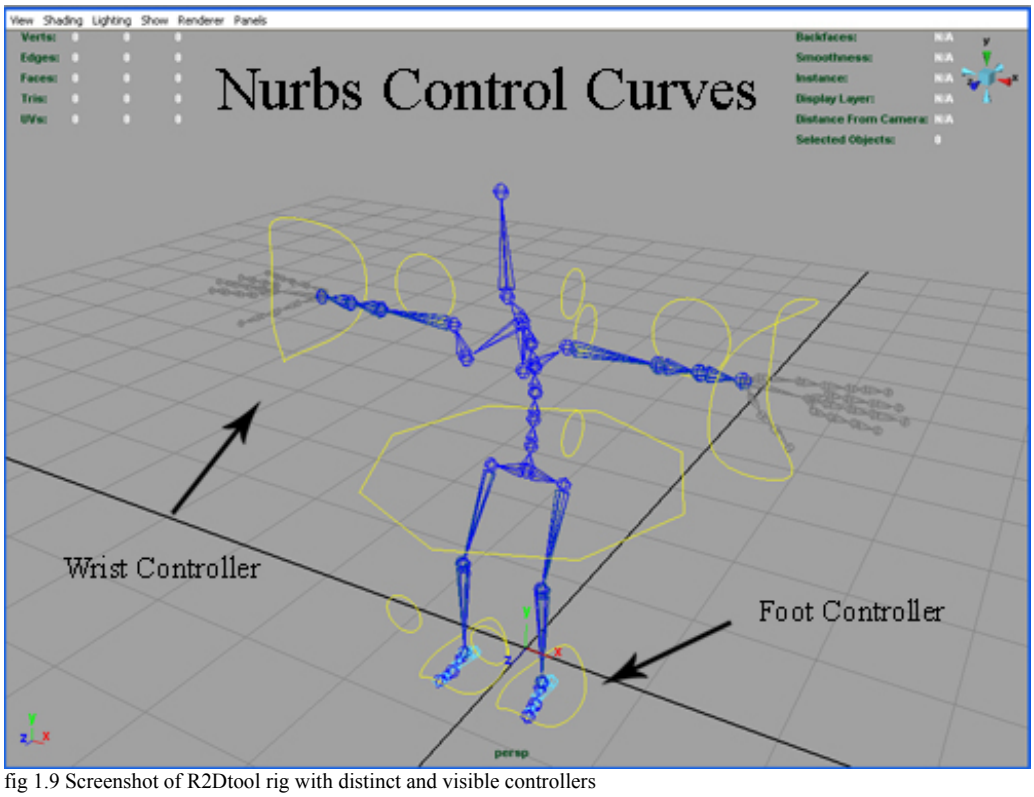

fig 1.9 Screenshot of R2Dtool rig with distinct and visible controllers

- **Clustering Curve Vertices**
  The usefulness of clustering curve vertices is best illustrated with the example of a specialised IK skeleton chain, which uses a curve to drive its transformations: the IK Spline. This is used in a biped character when rigging the spine. Deformations can be driven from the vertices themselves, requiring individual selection and keying. The coordinates of each vertex must be recorded, if the spine is to be returned to its start shape. These problems are avoided by 'parenting' Clusters to the various vertices on the spline curve as desired. Each Cluster retains information about it's own relative position (In Maya, this is known as a 'transform node'), and returning to this position is trivial. [fig 2.0]

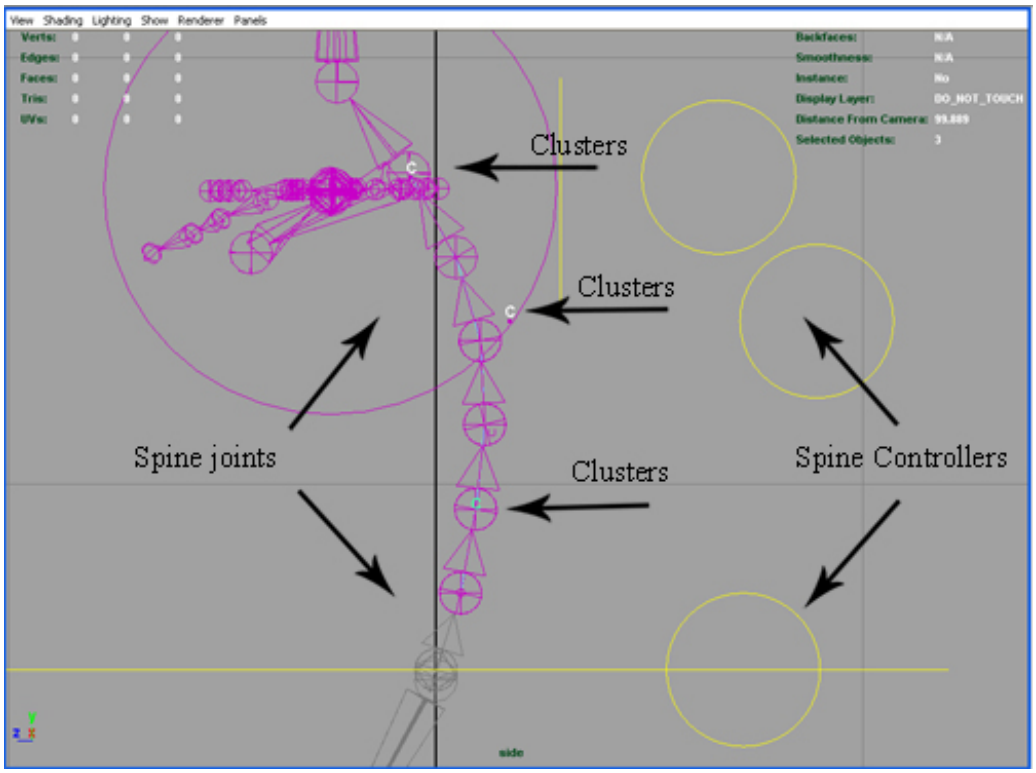

fig 2.0 Screenshot of R2Dtool rig showing use of clusters associated to nurbs curve controllers to control spine movement

- **Joint Orientation**
  To improve the animation pipeline, it is useful to keep FK joint orientation as uniform as possible. When applying IK splines to a joint chain, random local rotation axes (i.e. not aligned with each other) can cause the skeleton chain to react in an unpredictable and undesired way. [see f ig 2.1]

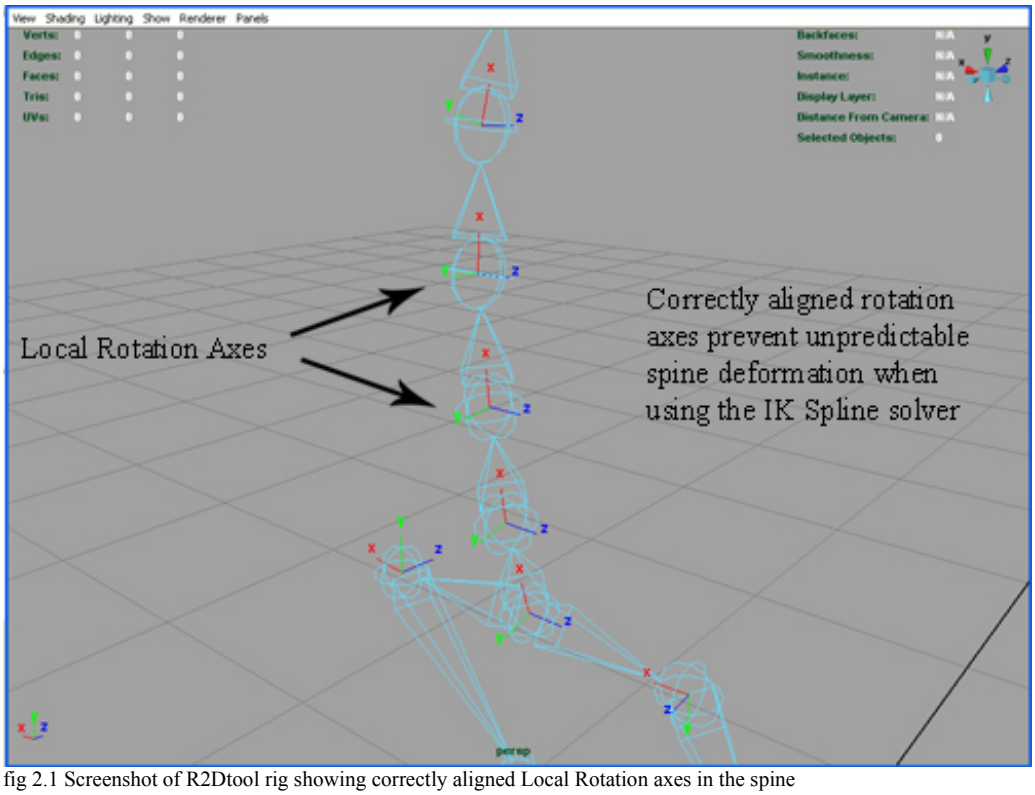

fig 2.1 Screenshot of R2Dtool rig showing correctly aligned Local Rotation axes in the spine

- **Additional Goal objects to smooth IK chain movement**
  Using the example of a tail, or tentacle, the addition of IK handles to various sections of a skeleton chain, can greatly improve the smoothness of movement between them as opposed to using just one.

- **An FK / IK Blended system**
  This can be achieved either by using a single skeleton joint and interpolating between the FK transformation of the joint hierarchy, and the IK solver solution, or by adopting a multi-resolution approach (see above), and blending an unrigged skeleton chain between the two solutions. Having an FK/IK blending facility to a rig adds considerable versatility, and allows for smooth changes of movement between Forward and Inverse kinematics. A good example of use would be in animating a character that catches a ball (IK) then smoothly throws it back (FK) [see fig 2.2]

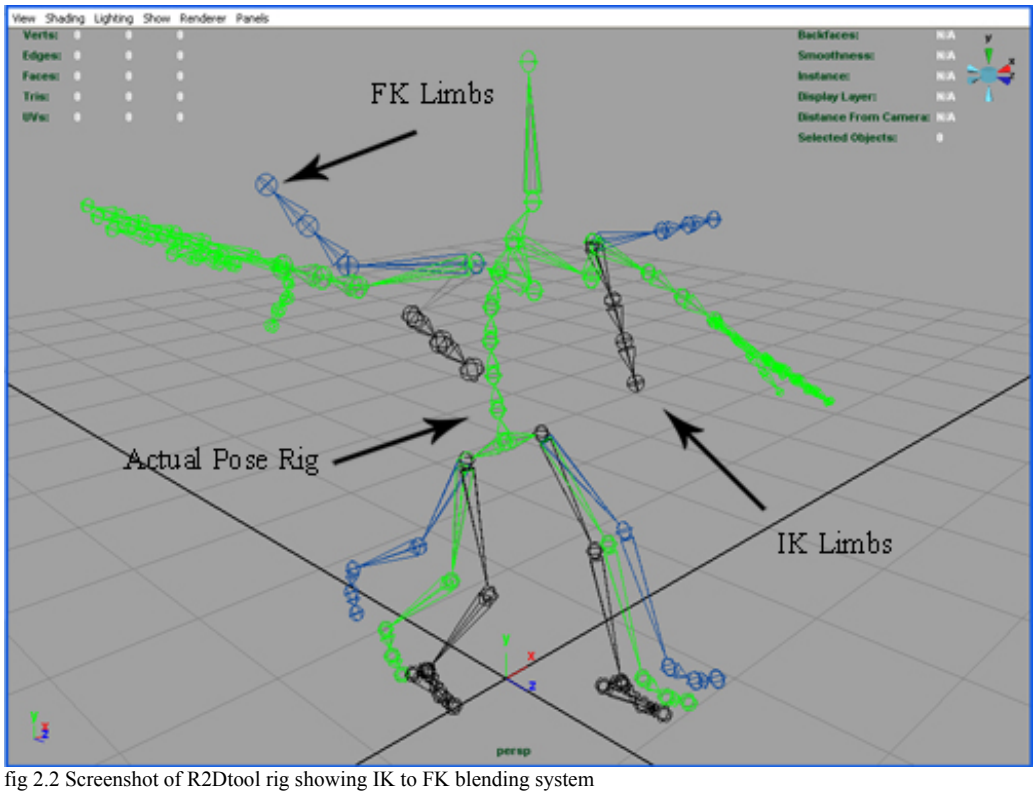

fig 2.2 Screenshot of R2Dtool rig showing IK to FK blending system

- **Reverse foot rig**
  Commonly found on modular rigs, the Reverse foot rig, provides essential 'natural' foot movement (including heel, ball and toe roll on any desired axes). The set up is fairly straight forward, but a little counter intuitive. [see fig 2.3]

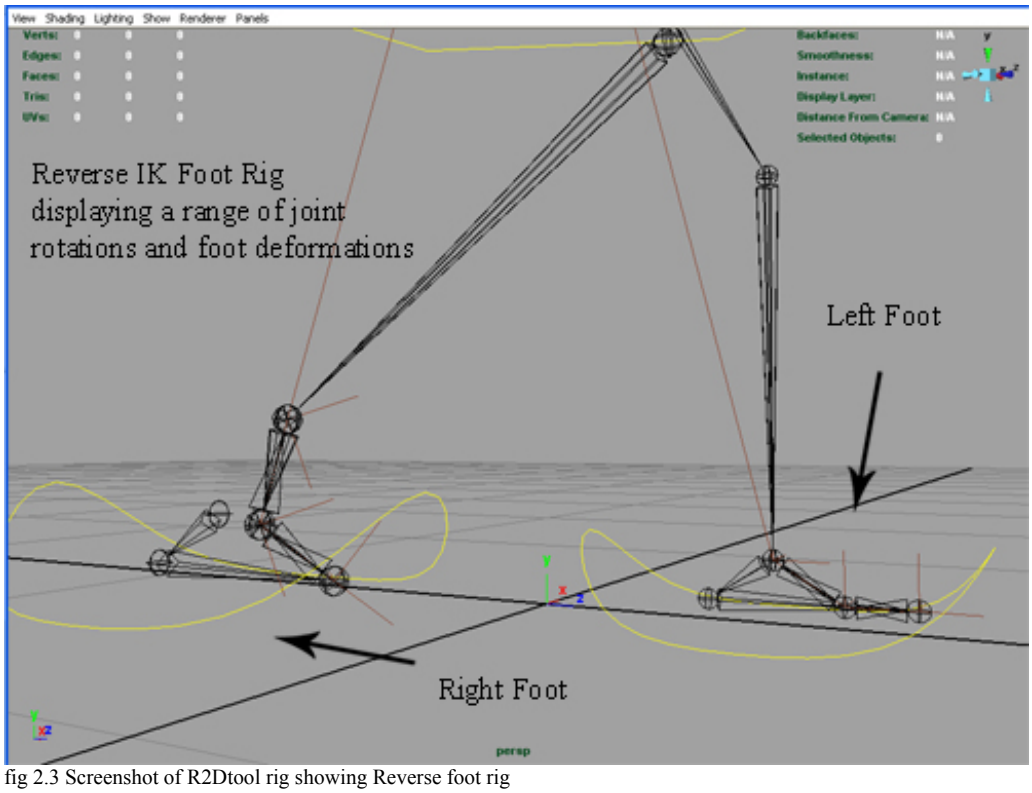

fig 2.3 Screenshot of R2Dtool rig showing Reverse foot rig

- **Modifying Joint pivots**
  Joints can influence movement, when the joints themselves are placed away from their own centre of rotation, or pivot point. This is best described with the example of rigging the 'eye' area.
  Positioning the joints for the top and bottom eyelids of a mesh, for example, and offsetting their own centres of rotation to match that of the centre of the eyeball . This removes the problem of the lids intersecting the eye when opening or closing, and instead they appear to follow the contours of the eye.

- **Automating Floor Contact**
  Adding this facility to a rig prevents the limb extremities from 'passing through' an elected surface, instead reacting to it by bending the immediate skeleton chain, once contact has been detected; quite simply, feet and hands will not pass through a floor object (provided this surfaces have been selected as 'floor contacts').

- **Swapping deformers**
  A number of deformers can have an affect on a skeleton chain, and blending between them can aid in creating effects evident in secondary, or 'overlapping 'animation. An example of this could be an IK rigged arm skeleton, with a dynamic curve) exerting control on some of the joints (In Maya, this is using the Hair system which can be used to produce a curve that deforms as a direct result of simulated physical influences, such as gravity).

R2Dtool

Swapping over to the dynamic curve would allow for the wrist to react to gravity rather then the IK solver.

- **Constraints and Rotation issues**
  Due to the sheer number of joints and constraints involved in even a simple rig, and the growing complexity that additional functionality adds (as described in this section), as a working principle, it is advised to rotate joints and constraints about their local orientation as opposed to global rotation.
  The significance of this becomes evident, when rotating an arm at the shoulder, but still requiring the elbow to rotate back toward the arm, whatever the shoulders final transformation; relying on global values would mean the elbow would now rotate in an 'unnatural' fashion.

1.4 Automating the process

'Look! It's moving. It's alive. It's alive... It's alive, it's moving, it's alive, it's alive, it's alive, it's alive, IT'S ALIVE!' – Dr Henry Frankenstein

"Frankenstein ", (dir) James Whale 1931

Image from James Whales 1931 film adaptation of Mary Shelley's' "Frankenstein"
http://www.theaterwissenschafterlangen.de/images/labor_Frankenstein.jpg

'Breathing life' into and animating a bipedal character to effectively create the 'illusion of life', involves a highly honed skill; it requires an appreciation of movement, weight, a precise sense of timing and an impeccable eye for detail. Animation is, by it's very nature, a time consuming art.
Character Rigging, as shown in the previous sections, can become incredibly complicated very early on into the process and errors in the construction of the rig may not be immediately evident. Much like animation, it is also a time consuming process.
It is on this premise that 'auto-rigging' tools have been developed, effectively automating the 'precarious' steps involved with the rigging process, but still allowing the user to cater the skeleton to the desired dimensions. Below is a brief look at some auto-rigging solutions.

R2Dtool

"Auto-Rigging Tool with emphasis on ease of initial set up, and incorporating a 2D limb placement"

- **MAYA FBIK –** ships with MAYA 7.0 or later

  MAYA version 7.0 shipped with a new rigging and animation system, from a previous software that had been acquired by MAYAs current owners, Alias. Coming from Kaydara Motionbuilder, the facility allows for Full Body Inverse Kinematics (FBIK), to be automatically applied to a skeletal structure that adheres to the required joint naming convention. Instead of IKhandles on each limb, FBIK effectors are moved throughout the body, to attain the various poses, causing a response from the other body effectors, and causing the skeleton to react in a 'natural' way. For example, bending over to pick up an object will result in a respective bending in the knees and back. [see fig 2.4]

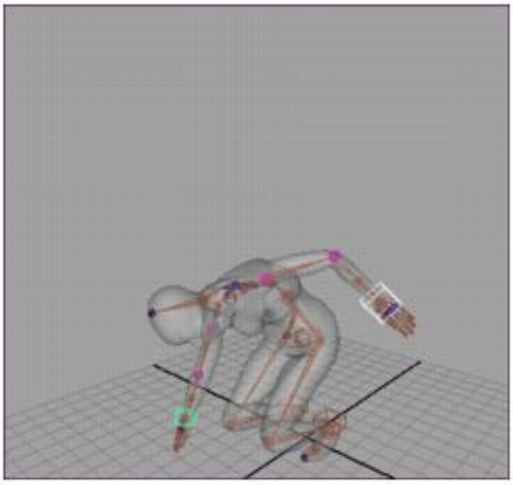

fig 2.4 Image from MAYA standard documentation, depicting FBIK movement in a character picking something up

- **CAT -** http://www.catoolkit.com/home.asp

  CAT is a 3D character animation plugin tool for 3DS Max, developed by SoftImage. Currently at version 2.5 at time of writing, CAT offers a wide selection of prebuilt modular rigs, which can be modified to fit the mesh of your character. These prebuilt rigs can have parts removed and added, or even form part of a brand new rig designed by the user.
  The interface works by placing 'pseudo-geometry' into the scene that displays joint hierarchy behaviour. This is then resized to fit the desired mesh proportions. As the rigs parts are modular, joints can easily be added or subtracted.
  CAT displays a feature similar in principle to MAYAs FBIK, called Effect Hierarchy. Joint positioning, whether in IK or FK animation mode, will cause bones in close proximity to rotate to 'accommodate' the move. [see fig 2.5]

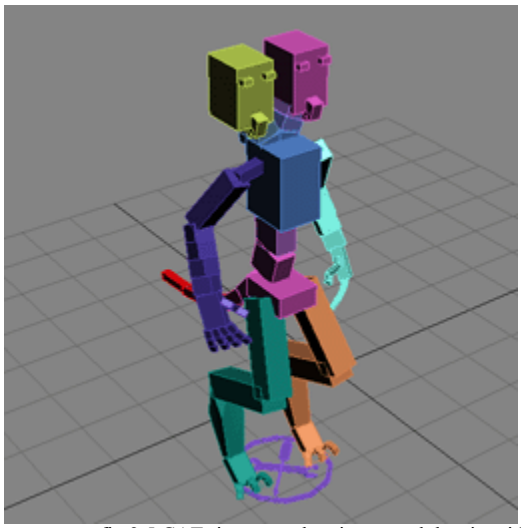

fig 2.5 CAT rig set up showing a modular rig with various rig body part

- **Final Rig -** http://www.radiantsquare.com/

  Final Rig, is a MAYA plug-in that automates and greatly reduces the time of the character set-up process. It has a modular approach to rigging, and allows the integration of other tool sets and rig designs to assist in controlling the character. Some of it's main features include: Multiple character sets, character proportion export functions, different limb set ups on the same rig, automatic Ik to FK limb blending and support for asymmetric character designs. [see fig 2.6]

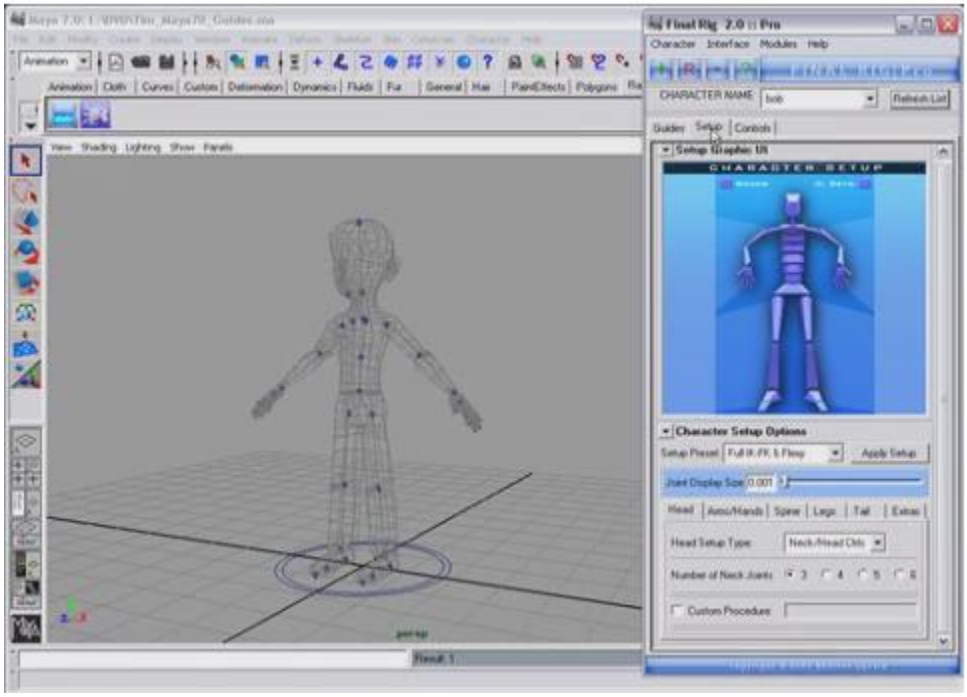

fig 2.6 Image showing the Final Rig v2.0 MAYA plugin tool
http://img157.imageshack.us/img157/2745/screenshot3783mj.jpg

2. Objectives

Having looked at other Auto-rigging tools, discussed functionality with industry personnel and with animators, the principle aim of this project is to design and produce an Auto-Rigging tool with an emphasis on ease of initial set up; this means an ‘easy method of rigging’.

The rig itself will incorporate essential features with an investigation into the possibility of adding Dynamic animation.

The reasons for adopting this project are to simplify a complex procedure and in the process, perhaps create a novel and ‘intuitive method’ for rigging; the point of departure for this will be a simple 2D method for the joint placement in a rig.

3. Implementing ‘R2Dtool, an automated 2D driven 3D rig tool

3.1. A 2Dimensional element to the placement of ‘joints’

‘Everything should be made as simple as possible, but no simpler’
Albert Einstein

The most obvious strategy for an easy and intuitive placement of joints in a 2D environment is to place markers at the desired locations; this immediately presents us with a multitude of problems.

- Creating a 3D structure from only 2D information
- Placing many closely positioned and partially overlapping joints in certain complex skeleton chains (e.g. the spine and the fingers)
- Creating a reliable system for limb recognition, when automatically placing joints
- Creating a usable and versatile rig with the previously explored functionality
- Using an intuitive method for marking the joints, and still being able to effectively retrieve the information

The method chosen for this study is the simplest possible; the physical placing of coloured dots, to mark certain 'key’ joint locations, on to a sheet of paper with an illustration of the desired proportions of the rig. This is effectively creating a rig simply from the design sheet of a character.

Utilising both the front and side views of a character design, the XY coordinates from the front view, and the YZ coordinates from the side view, can be combined and averaged to generate relative XYZ coordinates for any given point on the character and any subsequent rig.

Bearing in mind the problems stated above and taking into account the requirements of a functional bipedal rig (see section 1.3), a simple ‘colour – joint’ convention is employed, allowing for the indication of most of the major joints required to build a simple rig. Additional joints are added later on in the process.[see fig 2.7]

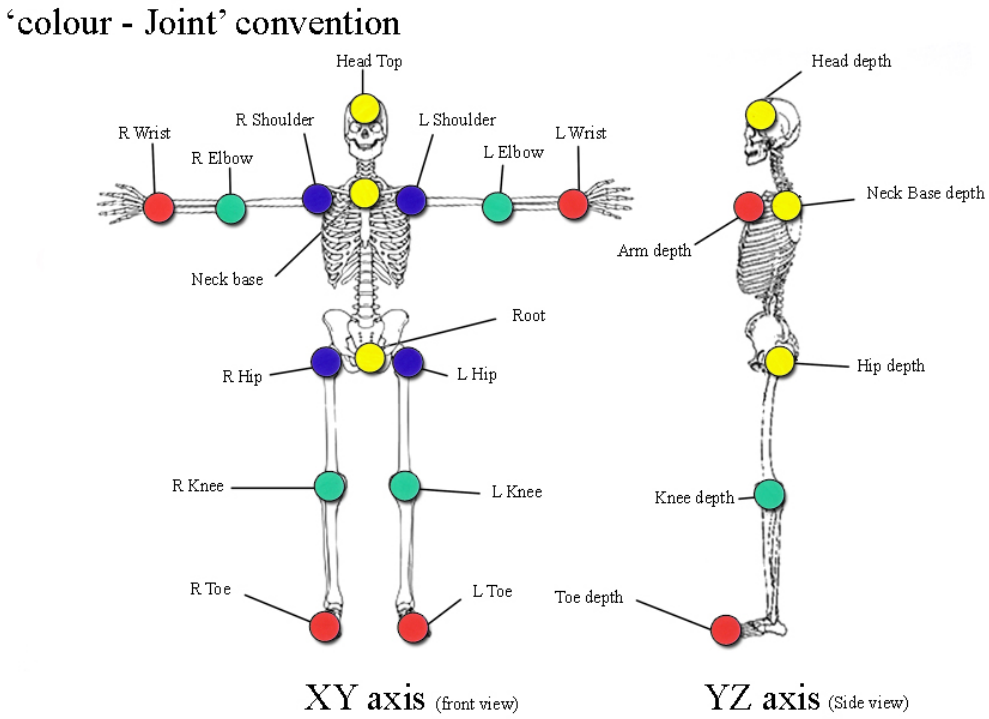

fig 2.7 Image showing the proposed 'colour-joint' convention

The information in the image is obviously required in a digital format, in order to implement the auto-rigging tool. Therefore, once it has been scanned, it is read by a separate program, that generates the coordinate details. The auto-rigging tool then reads these, and a rig is produced. The image processing part of the auto-rigging tool is a short Python script utilising the Python Image Library (PIL) (see section 3.2).

3.2. The Python solution

'The Python programming language manages to reconcile many apparent contradictions: it's elegant and pragmatic, it's both simple and powerful, it's very high-level yet doesn't get in your way when you need to fiddle with bits and bytes, it's suitable for programming novices and great for experts too,'

"Python in a Nutshell", Alex Martelli , 2006 [3]

3.2.1 Choosing Python

R2Dtool

"Auto-Rigging Tool with emphasis on ease of initial set up, and incorporating a 2D limb placement"

Python development began in 1990, by Guido van Rossum. It is a very high level programming language (VHLL) can work cross-platform on all major operating systems, is object orientated and most importantly it is very stable.
With standard libraries capable of accessing most elements of an operating system, a multitude of 3rd party modules, and a simplistic syntax, it is an excellent choice for software development.

The release of MAYA 8.5 includes full Python integration for the first time, further supporting the use of the Python scripting language for this study.
There are various utilities to create standalone applications for both Windows (Py2exe) and Linux (freeze.py) operating systems, meaning that cross platform distribution is also a straightforward process.

The image processing programme that reads the 'colour – joint' dot information for the positioning of the rig joints, has been developed using the latest Python release at the time, version 2.5. Despite an impressive array of libraries, Python 2.5 does not ship with adequate image processing facilities for the purpose of reading and manipulating pixel information; this problem is resolved, however, by using the Python Image Library (PIL).

3.2.2 The Python Image Library (PIL)

> 'This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities. The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.,'
>
> **"Python Image Library handbook", 2005 [4]**

The Python image library (PIL) supports many file formats and is 'ideal for image archival and batch processing applications'. Many GUI (Graphical user Interfaces) toolkits provide support for PIL should a user interface for the auto-rigging tool, be designed and implemented at a future date or expanded to incorporate new features. At the time of writing, the Python Image Library is at version 1.1.6

3.2.3 The Image Processing Programme

A transcript of the code can be found in appendix I, but for the purposes of simple illustration, the process by which coloured dot coordinates are obtained is shown in the 'Image Process Program' flow diagrams in appendix II. A basic overview of the process is described in appendix III. The resulting file from this program holds a list of coordinates that correspond to the joints that will be placed in the scene. [see fig. 2.8]

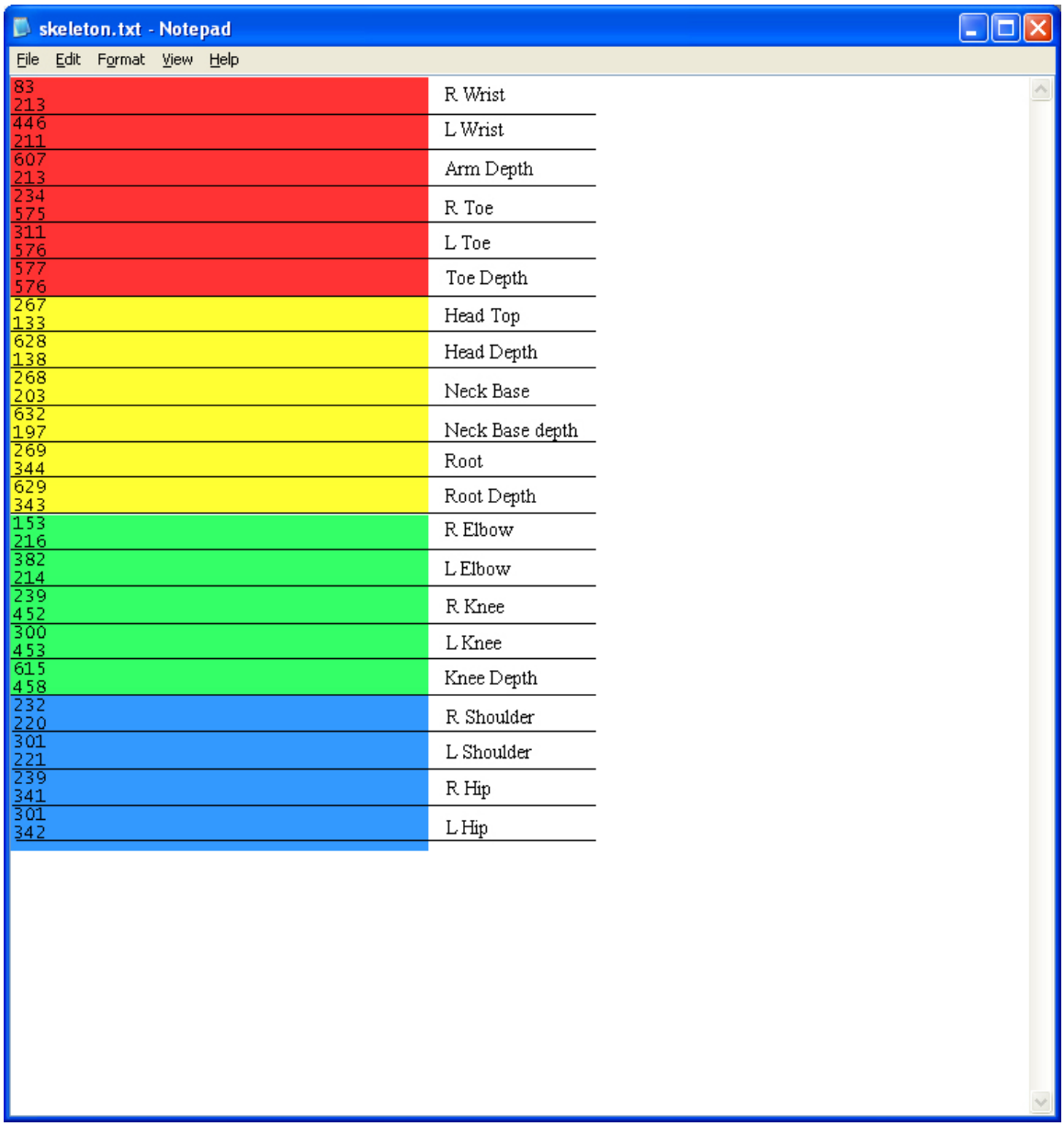

fig 2.8 Image showing the resulting skeleton.txt file from the Python Image processing program. The colours, lines and joint descriptions have been in this image for greater clarity. Each pair of consecutive values represents an X and Y coordinate, respectively.

MEL script for Rig Automation

3.3.1 Overview of MEL

'MEL is an acronym for Maya Embedded Language. This is the custom programming language designed specifically to work inside Maya'

"Complete Maya Programming", David A Gould, 2003 [5]

R2Dtool

"Auto-Rigging Tool with emphasis on ease of initial set up, and incorporating a 2D limb placement"

The ability to 'customize' MAYA in order to automate various tasks, create tools, accompanying GUIs and to drive procedural animation makes it an incredibly powerful and versatile 3D software package.
MAYA can be programmed using it's own interpreted language, MEL (Maya Embedded Language). Due to it's relative simplicity, and the fact that as an interpreted language it can be 'run' without the need for compilation, MEL allows fast and easy access to many aspects of MAYA that cannot be reached through using the standard GUI alone. The disadvantages of using MEL, in comparison to a programming language like Python or C, lie in speed of execution (especially noticeable with larger scripts) and very limited 'low level' system access.

Although the auto-rigging tool in this study was prototyped in MEL, with the new MAYA 8.5 release and full Python integration, it can easily be adapted and written fully in Python, speeding up the execution of the tool, and allowing for greater scope of functionality.

3.3.2 The Auto-rigger script

Due to the complexity of automating the rigging process and sheer size of the resulting script, a full copy of the MEL script developed for this study can be found in appendix IV.

The rig design itself is based on the basic steps and instructions presented in **"Character Rigging and Animation"[6]** , with extra functionality added from concepts found within **"The art of Rigging – Vol 1"[1]**, and additional modifications to achieve desired functionality (see section 1.3).
The resulting product is a 'Multi-resolution bipedal rig' with IK/FK blending on all limbs, scalable modular hands and feet and a facility to blend to dynamic animation (see section 1.3.1). The 'Squash and Stretch' ability, although investigated, has not been included in this study.

Diagrams depicting the flow of the process can be found in appendix V, and a basic overveiw of the process can be found in appendix VI.

4. Evaluating the Tool

Several examples of character designs were tested with the R2Dtool. Listed below are the steps required to implement the tool, and a display of the results.

1. Obtain Front and Side images of the character design to be rigged. These may be scanned images, photos, or model stills.

fig 2.9 obtaining the image

2. Using an image editing package, place the Front and Side images side by side on the canvas (preferably at equal spacing).

fig 3.0 Getting front and side views together

3. It is important to remove any colour saturation at this point. Leaving colour in the image may cause the Python Image Processing program to behaviour unexpectedly, producing random results

4. Either print the image off, and place the dots by hand, ready to rescan, or using an image processing package, create the coloured dots. The image to be used must be A4 sized at a resolution of 72dpi; this makes for image dimensions of 867 x 637. The coloured dots, when scanned at this resolution are 24 pixels in diameter. Placing dots using an image manipulation program must also be this size.

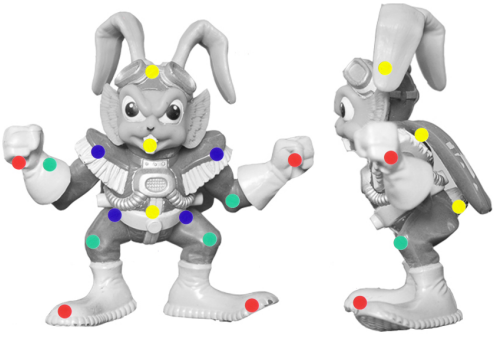

fig 3.1 Adding the coloured dots (see fig 2.7), to the desaturated image

R2Dtool

"Auto-Rigging Tool with emphasis on ease of initial set up, and incorporating a 2D limb placement"

5. Produce a .jpg version of this image and place it in the R2Dtool script directory
6. Load MAYA and load the R2Dtool script
7. Define the script, then run the following command from the MAYA Command line:

   R2Dtool("*image_name* ",*symmetry*)

   Replace *image_name* with the name of the image to be rigged from. Replace *symmetry,* with either 0 or 1 depending on whether an asymmetric rig is desired or a symmetric rig, respectively

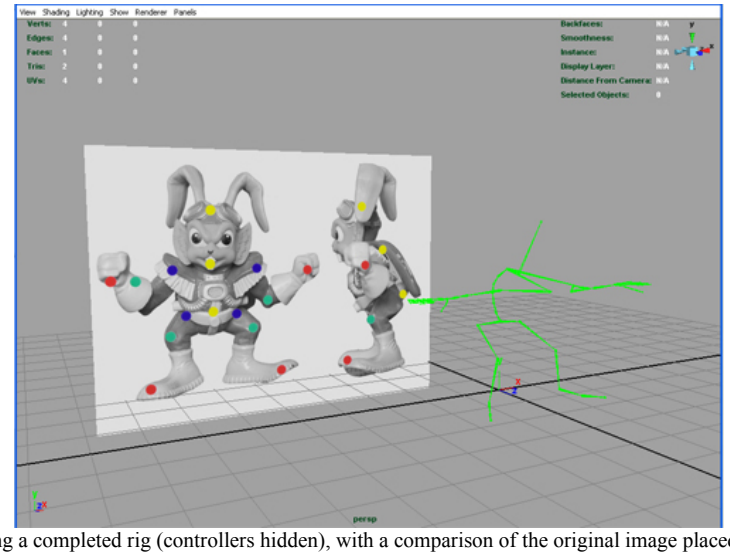

fig 3.2 image showing a completed rig (controllers hidden), with a comparison of the original image placed on a plane set to the same dimensions as the image. The resulting scale is 1:1.

5. Critical Analysis

The R2Dtool auto-rigging script works very well rigging 'T-posed' designs with a multi-resolution rig and dynamic animation blending facility, however there is still much development needed in terms of distributing this as a usable tool.

The tool has no user interface, so commands must be typed in; nor does it provide an intuitive access to the control of the rig. Investigating and developing solutions to these issues would require individual projects themselves, and are beyond the scope of this study.

5.1 Functionality

Functionality built into the auto-rig includes the following:

- IK / FK blending on all limbs
- Reverse Foot Rig
- Dynamic Animation blending
- Modular Hands and Feet that can be independently scaled

5.2. Assessment

5.2.1 "Placing the joints"

Acquiring the joint coordinates from the 2D image, is quick, effective and accurate, but the process of printing off a 'colour desaturated' image and manually placing coloured dots onto it can be quite a tedious task. However the physical act of placing these 'markers' is arguably the simplest method of joint placement and this aspect of interaction must be considered for future improvement on this process. Current thoughts are to implement this with a plugin or standalone program that displays the images on the screen so the user can 'drag' joint makers into place (much like stringing a puppet in 2D). This would improve accuracy and speed up the entire process.

The system of testing for colour, is also highly dependent on the actual coloured dots used in the study. To change these dots, would require altering the Python script.

Greater accuracy in joint placement can also be achieved by enhancing the method by the dots are examined. Optical Character Recognition could be

employed to read numbers or letter, in tandem with the colour scheme.

5.2.2 "Building the Bones"

Basing the placement of additional joints without coordinates, on the joints that already exist (e.g. position of the ankle inferred from the knee and toe joints), makes many suppositions on physiology and user requirements. The process has to be fully interactive, so the user can correct any automatically placed joints that do not 'fit' into the desired skeleton.

Calling the R2Dtool script with a symmetry setting of 1, leads to an averaged, symmetrical rig, which usually performs more predictably at the expense of inaccurately representing the image from which it was built

There are obvious limitations with the placing of joints from the coloured dot system used. Rig variation can easily be obtained from the front view, but the side view does not allow for differences in 'depth' of the limbs. The way around this would be to use OCR (See above), warranting further study on this topic.

At present the R2DTool auto-rigging tool automates the full skeletal build, but in adding a GUI to the tool, it should be fairly trivial to allow the user to engage at various step of the process, to improve the end result.

5.2.3 "Moving the Rig"

The rig performs well, from T-posed character designs. The IK/FK blending feature is predictable and stable on all limbs. The modular hands and feet can be successfully rescaled even after the rig is complete, but the hands need to manually aligned by the user once the process of auto-rigging has finished.

5.2.3 "Going Dynamic"

Blending IK/FK animation to imported dynamic animation also works as intended, but the entry and exit poses to the dynamic portion of the animation must be closely matched by the IK/FK pose; if the dynamic animation begins with the character in a prone position, and IK/FK rig pose is 'standing', a blending action will result in a linear interpolation of poses, and an unnatural deformation of the rig will occur, as the joints find the quickest, straightest route to their parent joints. There are problems with the orientations on the modular hands, during the blend to dynamic animation. This requires further study to resolve.

5.3 Expanded Knowledge base

R2Dtool

"Auto-Rigging Tool with emphasis on ease of initial set up, and incorporating a 2D limb placement"

Implementing the R2Dtool auto-rigging project has required study in MEL script, Python, program design and Auto-Rigging practices and techniques; this has lead to a necessary improvement of vocational skills.

6. Project conclusion and after thoughts

My intention for the 'Innovations unit' was always to try something 'challenging'. There

seemed little point in attempting an 'easy' subject, and nothing would be learnt from doing something I could already do.

Five months later and I've made a script that can rig from the 2D placement of joints, by sticking little coloured dots on a picture.

It sounds crazy but it actually works; it's not perfect and needs further development, but reading over my project proposal brief, I am satisfied that I have achieved all that I set out to achieve, and most importantly, I have attempted a difficult and technical project with a degree of success; however, whilst the original project idea was innovative and original, I feel I am left with a slightly 'empty' feeling, and this intrigues me. I'll explain why:

With this project, I have felt as if I have been on a continual 'back footing', that is to say, with the hours put in to the project, I feel as if I have relatively little to show and I believe this lies in the desire to do a 'complete' job; although my brief doesn't actually state this as an intention.

Perhaps this feeling is due to the 'fledgling' tool I have created, which is neither a painting, nor a model, nor an animation, but something that works for a few seconds and produces a mere framework to 'create movement'. In thinking this way, I am lead to an uplifting thought, and reminded of exactly why I am at university: I am here to learn new things.

Over the course of the Innovations unit, I have learnt the basics of Python script, learnt MEL to a much greater degree than before, investigated modern approaches to rigging and learnt to just keep pushing and pushing until a resolution was achieved, even when all motivation and confidence was gone (due, in part, to the Python Image processing program not working for a long time and the pressures of the other projects). Progress through the project, has not been linear, and I have felt at times of abandoning the whole idea, but 'chipping' away at the smaller problems eventually resulted in the desired outcome.

In as much as I have learnt new vocational skills, I have also learnt to bring a fairly expansive project with no 'right or wrong answers' to a successful conclusion in terms of a preset brief, resulting in a usable product.

7. Appendices

- Appendix I - Python Source Code for Image Processing Program
- Appendix II – Image Processing Program process overview diagrams

- Appendix III – Image Processing Program process steps
- Appendix IV - MEL Source Code for Auto-rigging script
- Appendix V - Auto-rigging script process overview diagrams
- Appendix VI - Auto-rigging script process steps
- Appendix VII – Coloured dots example

8. References

R2Dtool

"Auto-Rigging Tool with emphasis on ease of initial set up, and incorporating a 2D limb placement"

[1] Ritchie, K., Callery, J., Biri, K.,2005. *The Art of Rigging – Vol. 1*. San Francisco: CG Toolkit.

[2] Pelican, K.A, 2007. *Body Movin – Tips and tricks*. 3D World. Bath: Future Publishing Ltd, (2), pp.46-50

[3] Martelli, A.,2006., *Python in a Nutshell*. 2nd ed. Sebastopol: O'Reilly Media, Inc.,

[4] pythonware.com., 2005 *Python Imaging Library Handbook* . 1.1.5 ed. Available from: http://www.pythonware.com/library/pil/handbook/index.htm [cited 12 March 2007].

[5] Gould, D.A.D.,2003. *Complete Maya Programming*. San Francisco: Morgan Kaufmann publishers

[6] Alias Wavefront, 2004. *Learning Maya 6*. U.S: Sybex Inc.

9. Acknowledgements

Thanks to Ari Sarafopolous for his tutorage, Steve Harper for his perspective, Luke Titley for his logic, Chris Roberts for showing me that my simple ideas were actually better than his stupidly complicated ones, Steve Bell for inspiration.

Special thanks to Karl Wickens from EA and Phil Warner from CodeMasters, for their input and advice.