

Miami International University of Art and Design

A thesis submitted in partial fulfillment  
Of the requirements for the degree of  
Masters of Fine Arts in Computer Animation

Kinematically Speaking

Juan Borrero  
Winter 2008

Approved by:

---

Tom Joule, MFA Chair,  
Computer Animation

---

Alvaro Sanint, MFA Thesis  
Committee Chair, MFA Professor  
Of Computer Animation

---

Ahmed Shehata, MFA Thesis  
Committee Advisor, MFA  
Professor of Computer Animation

---

Gema Naredo, MFA Editor/  
Professor of Computer Animation

I grant Miami International University of Art and Design the non-exclusive right to use the work for the purpose of making single copies of the work available to the public on a non-for-profit basis, if the University's circulation copy is lost or destroyed.

Date: \_\_\_\_\_ Signature: \_\_\_\_\_  
Name: Juan Borrero

Miami International University of Art and Design

This dissertation, written by Juan Borrero, and entitled “Kinematically Speaking”, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

---

Alvaro Sanint  
Committee Chair, MFA Professor  
of Computer Animation

---

Gema Naredo  
Editor/Professor of Computer Animation

---

Ahmed Shehata  
Committee Advisor, MFA Professor  
of Computer Animation

---

Tom Joule, MFA  
Chair of Computer Animation

Date of Defense: March 13, 2008  
The dissertation of Juan Borrero is approved.

---

Deborah Mass  
Dean of Academic Affairs

---

Tom Joule, MFA  
Chair of Computer Animation

AI MIAMI INTERNATIONAL UNIVERSITY  
OF ART & DESIGN

**KINEMATICALLY SPEAKING**

A THESIS SUBMITTED TO THE GRADUATE SCHOOL  
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE

MASTERS OF FINE ARTS  
COMPUTER ANIMATION

BY

JUAN M. BORRERO RIVERA

©2007 Juan Borrero (Dragonlord)

## I. Table of Content:

i.	Table of Content	2-4
ii.	Copyright Notice	5
iii.	Acknowledgements	5
iv.	Vita	6
v.	Introduction	7-16
	a. Conceptual Development	7
	b. Overview of the Thesis	7-8
	c. Purpose of the Thesis	8-9
	d. Definition of Rigging	10-11
	e. Modeling for Looks or Animation	11-12
	f. UV Mapping	12-13
	g. Deformers	13-15
	h. Plug-Ins	15-16
vi.	Historical	16-18
vii.	Maya Work	18-35
	a. Process	20-24
	1. The Graphical User Interface	20-22
	2. Muscle Cages	22-24
	b. Current Version	24-25
	c. Software Limitations	25

d. Pipeline Workflows	25
1. Rigging Basics	25-26
2. Spine Setup	26-29
3. Blendshapes	29-30
4. Keeping History	29-30
5. Joint Orientation	31
6. Leaving a Service Entrance	31-32
7. Controls Basics	32-33
8. IK's Vs. FK's	33-34
9. Utilities	34-35
10. GOD Complex	35-36
viii. Tool Explanation and User Guide	36-41
ix. Conclusion	41-43
a. Conclusion	41-43
1. Limitations	41-42
2. Problems	42
3. Results	42-43
b. Appendix	43-68
1. Glosary	43-68
c. Documentation – Troubleshooting	68-70
1. Installation	68-69
2. Interface	69-70
3. Integration	70

4. Usage	70
5. Scripts	70-71
d. Works Cited	72
e. Works Consulted	72

## II. Copyright Notice:

All contents of this thesis belong to Juan M. Borrero and the institution of Ai Miami International University of Art & Design. Any reproduction of this piece of work has to be with written consent from the owner and all parties involved. cMuscleSystem is property of Autodesk Inc. cgMuscle is property of Judd Simantov and Mark Edwards. Muscle TK is property of CGTOOLKIT. All third party materials have been used for educational purpose and its contents have been left intact. 2008.

## III. Acknowledgements:

I Juan M. Borrero Rivera would like to thank the following people that in some way have helped me develop into the human being I am today, and the human being I'll grow in to be. These people were, are, and will be my guidance for this project and projects to come. These acknowledgements are in no particular order and are equally important:

Gema Naredo	Myriam Rivera	Juan Borrero Velez
Ahmed Shehata	Alvaro Sanint	Vanessa Velasquez
Idania Borrero	Dr. Barbara Falletta	Leonardo González
Tom Joule	Silvia Sayas	Gigliola Cikowski
Carmen Muñoz	Deborah Mass	Judith Anderson
Marco Garcia	John Carret	Gracie Head
Pedro Alvarez	Frank Velazquez	Craig Kelly
Lazaro Terrero	Melissa Camic	My Family
My Friends	SYNQ Animation Studios	

Faculty of Ai MIU of Art & Design

#### IV. Vita

I was born in Hato Rey, Puerto Rico on December 20, 1984. My parents are Juan B. Velez and Myriam R. Rivera. I received my secondary education at Colegio Espíritu Santo in Puerto Rico where I graduated with honors. I was interested in the arts and its technical aspects; therefore, I pursued a career in computer animation. In June 2003 I joined the Computer Animation program at the Ai Miami International University of Arts & Design, from which I graduated with a BFA degree in September 2006. During my studies I worked in conjunction with my teachers Alvaro Sanint and Ahmed Shehata in various projects. My main focus while working with my teachers was character rigging and deformations. I later went on to join their studio as an in-house rigger, where I developed my skills and acquired some field experience in management, networking, team leading, and resources management.

In October 2006 I was admitted to the Graduate School at Ai Miami International University of Arts & Design. My focus for joining the program was the creation of a new muscle deformation solution, and the ability to teach in a College level environment. I was granted the degree of Master of Fine Arts in July 2008.

#### V. Introduction

## A. Conceptual Development

Due to the researcher's interest in the technical aspects of animation and his eagerness to learn new techniques, he chose a project complex enough to motivate him to learn new material. With this in mind, his goal was to create a thesis in which his scripting and rigging skills would integrate and improve his field of study.

The primary goal of this project is to create a muscle deformation tool that could be used across all 3D software. This task, although challenging, posed a more sophisticated product than researching upon character rigging techniques. Juan was inspired by the tools that are already on the market and made sure that the tool which he was developing performed the same functions but eliminated the flaws. With the help of his teachers and co-workers he began to create a tool that is not only useful but can contribute to the body of literature in the field of computer animation. First, he had to familiarize himself with the base codes of the applications targeted and also learn the programming language of C++.

## B. Overview of the Thesis

For the purpose of this thesis the researcher attempted to create a virtual 3D muscle tool which he name *JBMuscleSkin*. In the creation process, there are many steps and challenges to be met. The general outcome of *JBMuscleSkin* is to create a character rigging workflow that takes less time, less computer resource dependence, and is more user-friendly than other tools on the market. *JBMuscleSkin* has an easy application setup as

well as some graphical interfaces to help facilitate the setup for both the technical department and the artist.

To accomplish these tasks, *JBMuscleSkin* will be written using programming languages that are native or recognizable by the software target markets. The software targets are Autodesk's 3D Max and Maya, and Avid's Softimage XSI. To create the interface, the researcher used the open source programming libraries of wxWidgets. The creation of OpenGL (Open Graphics Library) cages for 3D space representation of the muscle was used. These cages were accomplished using classes from the Maya API (Application Program Interface) knowledge base, and the C++ programming language.

This thesis includes some Maya deformer plug-ins that are additions to *JBMuscleSkin*. These Maya plug-ins include aspects of basic shape deformers to collision effects. Also, the muscle cages are setup as modular creations. In other words, no matter where the cage is set, the muscle will be created with as accurate dimensions as possible. Furthermore, the muscle cages will be created in a way that the user can reshape them to fit his character structure.

### C. Purpose of this thesis

Computer animation is quite young compared to other fields of study. Even though the discipline is relatively new, the volume and growth is extreme. The development of the Computer Animation field is intertwined with that of advancing technologies. Because of this the field is evolving,

and newer and more effective techniques and methods are created daily. This thesis is an attempt to enhance the process of developing the specialization of character rigging. The goal of this paper is to create a muscle deformation system that can be applied to all rigging solutions across the board minimizing arduous setups and the tedious paint weights process.

Character rigging is the computer animation process that is in charge of creating all character deformations. Many people outside the field might be confused about the true nature of character deformation and, in reality, it is very simple. Character deformation is the ability of a 3D character to move, interact, smile, frown, jump, etc. Moreover, character deformation covers the creation of virtual joints and bones, muscles, and organic matter. A jelly-like belly, a horse's muscle interaction within his anatomy, and even the way our skin moves and folds when we close our hands and extend and retract our arm, are examples of character deformation.

This proposal, is not trying to recreate what already exists in the field like "cMuscleSystem" and "muscleTK", which are predominantly successful tools on their own that achieve a close to real muscle system in 3D. This project will cover the majority of those scenarios, in addition to demonstrating a self contained virtual muscle creation tool that can be applied to all 3D character shapes and sizes.

#### D. Definition of rigging

Animation cannot take place unless the rigging process has finished.

Rigging is basically the strings to a puppet in a puppet show. This process is solely the one in charge of creating all the controls and mechanisms to move a character in 3D space. The Maya documentation files define it as:

*Rigging a character, also known as character setup, involves creating skeletons and IK handles for your characters, binding skins to the skeletons, and setting up deformers and constraints. You can also create deformers for your character and animate them to produce effects; for example, the jiggling belly (jiggle deformer), furrowing brow (wire deformer), and flexing biceps (lattice deformer) of a sumo wrestler model.*

*Non-character objects are also very important to bringing your scene to life. You can limit and control the transformations of objects by constraining them to characters or other models in your scene. You can also create deformers for objects to create complex deformation effects. For example, you can apply a squash deformer to the model of a ball and then parent constrain the ball to the hands of a character. With this setup, you can key the weights of the character's hands and the squash deformer's attributes to create an animation of the character bouncing the ball from hand to hand while the ball squashes on the ground and stretches as it rises back into the air.*

*In addition to setting up characters and objects for animation, you can set up Maya® Dynamics™ for animation. You can constrain dynamic*

*objects such as particle emitters, fields, and fluids to objects or characters in your scene.* (Maya Documentation Server, 2007)

To start rigging your character and to use the tool described in this thesis it is important to know some information about modeling and the rigging pipeline. The process of rigging is extensive and for this reason more in-depth information about the pipeline and the process can be read later on.

#### E. Modeling for Looks or Animation

In the field of animation riggers encounter many obstacles when dealing with models. Most of the time, modelers have the task of creating a character that looks a specific way but hinders the process of character rigging. In effect, it is important to know some of the common mistakes and practices in the character modeling department.

First off, the most common error in character modeling is lack of geometry. For example a model that does not have enough edge loops around the elbow, shoulder, wrist, amongst other bending areas. A lack of edge loops in these areas will result on loss of volume or collapsed shapes. To fix this problem it is recommended that rotation areas like these have from three to five edge loops to maintain the shape. The reason for the three loops comes from a simple fact, one edge is the defining line (center edge), the one above is the curvature tangent of the shape, and the last one is the falloff onto the next joint. Insufficient edge looping can be seen most of the time in game models, but is not always the norm.

Another way of having incorrect edge looping can be seen on models that are based off real characters. For example, if there is a game about “*Rugrats*”, the nickelodeon cartoon, the characters are already predefined and have to look a certain way. When a modeler is challenged with this type of task several times the looping becomes erratic to meet the overall design of said character.

Here is a list of things to keep in mind for both the modeler and the rigger when going over the geometry looping.

**Center Line:** The character needs to have an edge loop that goes down the center of the volume and is overall a straight line.

**Three-Five Rule:** All bending areas need at least three to five edge loops around the volume. These loops need to be perpendicular to the joints.

**Even Faces:** The paint weights process is less cumbersome when the polygonal faces on the characters are as even as possible.

**Three-To-One Fix:** When minimizing geometry looping it is ok to create a three-to-one edge connection in areas of little to no bending influence.  
(e.g. Forearm, Mid Shin, or Mid Calf)

#### F. UV Mapping

Before rigging gets underway there are a few steps one must keep in check. The first item is the geometry flow and the second one is the UV mapping. The later is important for one main rule, texture stretching. It is preferred to have these UV's done before the rigging process but is not a must. Having the UV maps done before rigging helps in determining the

texture stretching boundaries. Moreover, correctly mapped UV's help move along the pipeline process so that the texture artist and the rigger can work simultaneously without compromising the outcome of the work. As a rule of thumb one must try as much as possible to have the UV maps done before rigging. There are ways of applying UV's after the rigging process but their reliability is compromised because of software issues.

## G. Deformers

Maya deformers are those utilities inside the software that have been developed by an Autodesk(developer company) engineer or programmer. These deformers cover from blendshapes to Lattice deformers, non-linear deformations, and joints.

*“In Autodesk® Maya®, deformers are high-level tools that you can use to manipulate (when modeling) or drive (when animating) the low-level components of a target geometry. In other software packages, the terms modifiers and space warps are used to refer to what Maya calls deformers.”* (Maya Documentation Server, 2007)

Take notice that there are two main groups of deformers, the non-linear deformers, and the linear deformers. Their differences will be explained later on.

### 1. Usage

The use of Maya deformers ranges widely depending on the outcome of the product. For example, facial deformation could get accomplished

one of two ways, through the use of joints or the use of blendshapes. Both of them are effective methods but have their limitations and their benefits.

One very useful deformer is the lattice shapes. In the field it is used to facilitate the paint weights on heavy geometry characters. The application is very simple, select the vertices you want to get affected by the lattice shape (e.g. Torso) and create the deformer from the “Create Deformers” menu. After you have created the lattice shape, bind it to the joints you would have bound those vertices to. Paint weights on the lattice points with the “Component Editor” until you achieve the desired deformation.

## 2. Limitations

Maya Deformers have some known limitations that for the end user that doesn't know about them can hinder hours of work. One of these limitations is that deformers in Maya follow the hierarchy stack in which one affects the result of the next one in line. One such limitation is that of facial blendshapes being under the skin cluster stack.

Many of the deformers in Maya are good for low effects in which the uses of the twelve principles of animation are not needed. One such deformer is the squash and stretch deformers that lack the ability to being painted or tweaked beyond what is supplied with the software.

This leads to third-party deformer creations commonly called Plug-Ins.

## H. Plug-Ins

Maya Plug-Ins are third-party and Autodesk's developed tools that did not necessarily get shipped with the software. Some of these Plug-Ins are patches to fix problems or tools that enhance the work flow. The ranges for these Plug-Ins are in many areas, such as exporting, animation, deformation, and dynamic tools. These Plug-Ins are built using Maya's innate developing classes that are based off the C++ programming language.

### 1. Usage

The main purpose for Plug-Ins are essentially to enhance the software. They are located in the "Plug-Ins" menu under "Window --> Preferences". For them to appear here they have to be installed in the Plug-Ins folder under the Maya install directory. Most plug-ins come with installers and self extractors so this step is rarely necessary.

### 2. Developers

Plug-In developers for the Maya software range from market and volume. Some of these developers are individuals; others are groups and even corporations. The developer's scene is not that big, it takes time and effort to create a plug-in for Maya because of all the venues the software meets.

### 3. Limitations

Third Party Maya Plug-Ins have varying limitation levels. The first of these limitations is the fact that the users need to cater their work to

the pipeline of the developer of the plug-in. While many developers try to have their tools open for custom pipeline uses, sometimes their knowledge of certain pipelines limit the outcome of the product.

## VI. Historical

### **muscleTK: 2004**

First muscle solution for Maya. Even though cheap in price it requires a lot of computer resources. The Following is information from their website:

“Muscle TK is the world's first commercial muscle and skinning plugin for Alias Maya 5.0, 6.0, 6.5 and 7 on Windows and Linux.

CG Toolkit announces the release of Muscle TK for Alias Maya. Developed under the Alias Conductors program, Muscle TK gives creature TD's an alternative to traditional rigid and smooth binding techniques for their digital creatures. Muscle TK allows artists to give their creatures realistic musculature with two different methods of attaching that musculature to the creatures skin or 'mesh'. With Muscle TK skin, your digital creature's skin can realistically slide over muscle surfaces that are driven by underlying Maya joints. Maintaining your creature's body-volume while undergoing deformations has traditionally been a painstaking process involving influence objects and blendshapes. Muscle TK's muscle deformer

gives setup artists a quick and easy way to create fleshy body parts that squash and stretch in a realistic manner.” (www.cgtoolkit.com)

**cgmuscle: june 15, 2004**

Open source muscle solution for Maya. The solution is a community effort but the precursor founders of the site are Judd Simantov and Kark Edwards. The following is their first entry onto the sites main blog page:

“cgmuscle is a collaborative project between people in the industry to develop an open source muscle system for maya. This site is open for all discussion, but please keep in mind the goal here is to develop a muscle system, so try and keep discussion specific.

Also, at this point I'm not too sure yet on exactly how we going to carry this project out, so I think the operative thing to remember is that this is something for everybody. Which means any ideas, suggestions or help is more than welcome and would be really appreciated. I think keeping it organised and on track is definitely a priority. As much as Mark and I will be administering this, it's as much your project as it is ours. I think once we get enough people registered, we can all agree on a feasible way to carry out this project. On that note, register and flood us with your input please

One last thing, big thanks to Paul Thuriot who is currently working

at Tippett Studios and Jason Schleifer who is currently working at PDI as well as working on a short film that you can find here: [jonhandhisdog](#). Both these guy's were really the main inspiration behind this. That and my obsession with muscle systems

-Judd Simantov" ([www.cgmuscle.com](http://www.cgmuscle.com))

### **cMuscle System: 2005**

Due to Autodesk's purchase of the cMuscleSystem, [www.comet-cartoons.com](http://www.comet-cartoons.com) has removed all information pertaining to the creation of this tool. The tool was released in mid 2005 and took the field by surprise. The tool was totally integrated into Maya, it barely took any computer resources, and was to a certain degree user friendly.

## VII. Maya Work

This project was to create a tool named *JBmuscleSkin* which was to be integrated into the Autodesk Maya rigging workflow. Due to events that occurred in December 2007 this project will take a sudden change of outcome. On December 11, 2007 it was announced on Animation World Network that the *cMuscleSystem* created by Michael Comet was bought by Autodesk. This in effect came to the result of Autodesk integrating a muscle solution in their 3D package of Maya. These events made the outcome of this project somewhat useless, and hindered its result. *JBmuscleSkin* will not be the main focus of this thesis anymore and the focus will be more in tools that

help the deformation aspects of rigging and animation. Although the final product was already in development, further research will not be conducted for the outcome of this thesis. These tools will be an attachment to already known workflows and might enhance, but not guarantee, integration with the rigging tool better known as *Final Rig* version 2.3 developed by Ahmed Shehata, and Alvaro Sanint. Here's the article from the acquisition of cMuscleSystem from the Animation World Network Site and the Autodesk information website.

## **Maya Muscle Introduced for Autodesk Maya 2008 Extension 1**

**December 11, 2007**

Autodesk Maya 2008 Extension 1 software has been released, introducing Maya Muscle functionality. This comprehensive muscle and skin system, recently acquired from Comet Digital LLC, allows artists to create life-like skin motion through features that let them precisely direct muscle and skin behavior. The toolset integrates extremely well with the Maya architecture and overall workflow, enabling it to be used in isolation, interconnected with other Maya features, or customized and scaled as needed.

Maya Muscle provides technical directors and animators with the tools they need to create detailed skin articulation and animation. These include advanced muscle and skin sculpting and deformation tools, as well as extensive jiggle and weighting options, like slide, sticky and wrinkle weights.

In addition, Maya Muscle can increase artists' productivity by taking advantage of familiar user interface and workflow features included in Maya 2008, such as the software's brush-based interface. At the same time, it delivers a number of tools, including automatic rigs, realtime jiggle tweaking and file caching, designed specifically to make the process of achieving secondary character motion.

Autodesk Maya 2008 Extension 1 is now available for download to Autodesk Maya 2008 Complete and Autodesk Maya 2008 Unlimited Subscription customers with Gold support. The Extension 1 is not sold separately.

The SRP is \$1,999 for Maya 2008 Complete (Standalone) and \$6,999 for Maya 2008 Unlimited (Standalone). The upgrade SRP from Maya 8.5 Complete to Maya 2008 Complete is \$899, and the upgrade SRP from Maya 8.5 Unlimited to Maya 2008 Unlimited is \$1,249. Subscription with Gold support for Maya 2008 Complete (Standalone) is SRP \$1,295, and Subscription with Gold support for Maya 2008 Unlimited is SRP \$1,495.

For more info on Maya 2008 Extension 1, please visit [www.autodesk.com/maya-extension1](http://www.autodesk.com/maya-extension1). ([www.awn.com](http://www.awn.com))

### **Autodesk Maya 2008 Extension 1: Maya Muscle**

Maya 2008 Extension 1 offers the integrated Maya Muscle toolset exclusively to Autodesk Subscription customers with Gold Support – both those with Maya Complete and Maya Unlimited licenses.

#### **Create Life-like Skin Motion**

Maya Muscle delivers advanced muscle and skin deformation tools, jiggle functionality, paintable weighting and muscle sculpting options that give technical directors and animators the control they need to create life-like skin motion.

#### **Boost Your Productivity**

Maya Muscle takes full advantage of familiar, productivity-gear Maya UI and workflow features such as the software's brush-based interface. At the same time, it delivers a number of tools — rig presets, realtime jiggle tweaking, caching and more — designed specifically to make the process of achieving secondary character motion with Maya Muscle more efficient. ([www.autodesk.com](http://www.autodesk.com))

## **A. Process**

### **The Graphical User Interface:**

The biggest challenge in creating a virtual tool is the creation of its accompanying interface, typically called a GUI (Graphical User Interface).

A GUI determines the ease of use of a tool, its capabilities, and its complexity. Although a tool can have a simple GUI, the use of it might be quite complicated or it may not. For a programmer with little inclination

towards aesthetics and design, a GUI can be a challenging task. A GUI has to be pleasant to the eye yet functional and uncluttered. The GUI's for the tools developed in this thesis were made using Maya's MEL scripting language and some API.

The GUI's started with a window layout using MEL. The default window creation commands look as follows:

```
global proc JBmuscleSkin ()
{
    if (`window -ex JBGuiWin`)
        deleteUI JBGuiWin;

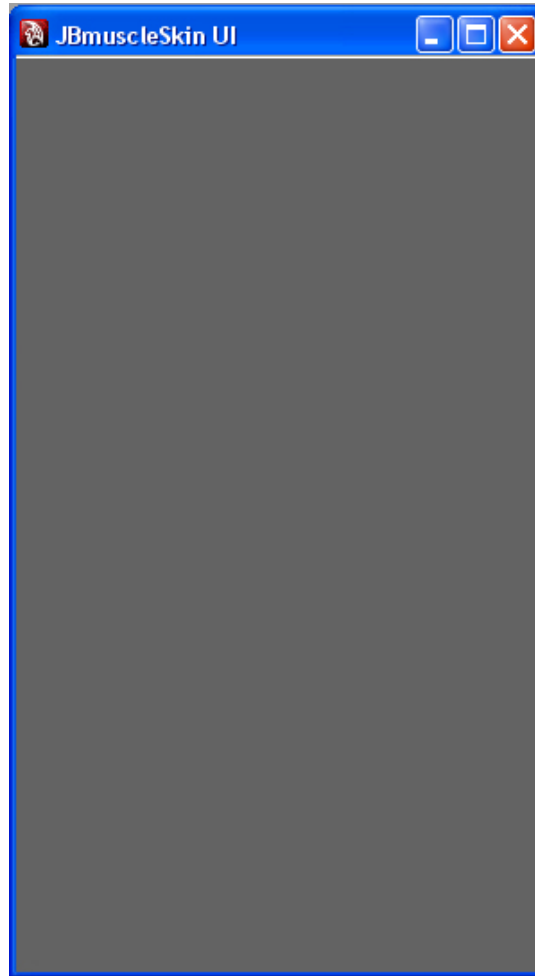
    if (`windowPref -ex JBGuiWin`)
        windowPref -e -w 300 -h 545 JBGuiWin;

    window -w 300 -h 545 -bgc 0.39 0.39 0.39 -mb 1 -tb 1 -t "JBmuscleSkin
UI" JBGuiWin;

    showWindow JBGuiWin;
}
```

The first line tells Maya that this window will be part of a Global Procedure in which any other script in the scripts folder can latch onto. The next four lines are "If" loop statements which tell Maya to check all the windows that are open. If they are named the same as our window, delete them, and clean the attributes of it's height so that it is always the height specified. The last two lines are the window creation command from MEL, a few specific attributes tell which will be the title of the window and how the user named his window in the script. The last line is simply telling Maya to

show the window after it was created in order to be used. The window made from these lines should look as follows:



As you can see, this window has a lower value than other windows, and it is empty. There is no functionality in this window and so this is where the window needs a face-lift.

There are a few layouts and functions in MEL that are going to be used.

These functions are the button controls, slider controls, option menus, shelf layouts, column layouts, form layouts, among others. These layouts, controls,

and commands will give the user the functionality needed to use these virtual 3D Tools.

### **The Muscle Cages:**

After researching various muscle setups, one can come to the conclusion that the way to start a muscle shape is from small to large. This project began by the designer creating very simple polygon spheres and plotting the points in space using API. In essence, these muscle cages and deformers are going to be Maya locator-type nodes that will hold information relating to the surfaces in which they interact.

The version of Maya initially used in this project was version 7.0, which has moved to 8.5 and even further to 2008 (9.0). These updates in the software base code created some obstacles for the outcome of the project which were based on the old libraries of the software. One problem encountered was the "mstatus.h" header file of the software. The problem consisted of some code that changed from Maya 7.0 to 8.5. The errors read as follows:

```
Compiling...
BasicLocator.cpp
c:\program files\autodesk\maya8.5\include\maya\mstatus.h(161) : error
C2653: 'std' : is not a class or namespace name
c:\program files\autodesk\maya8.5\include\maya\mstatus.h(161) : error
C2433: 'ostream' : 'friend' not permitted on data declarations
c:\program files\autodesk\maya8.5\include\maya\mstatus.h(161) : error
C4430: missing type specifier - int assumed. Note: C++ does not
support default-int
c:\program files\autodesk\maya8.5\include\maya\mstatus.h(161) : error
C2653: 'std' : is not a class or namespace name
c:\program files\autodesk\maya8.5\include\maya\mstatus.h(161) : error
C2061: syntax error : identifier 'ostream'
c:\program files\autodesk\maya8.5\include\maya\mstatus.h(161) : error
C2143: syntax error : missing ')' before ';'

```

```
c:\program files\autodesk\maya8.5\include\maya\mstatus.h(161) : error  
C4430: missing type specifier - int assumed. Note: C++ does not  
support default-int  
c:\program files\autodesk\maya8.5\include\maya\mstatus.h(161) : error  
C2805: binary 'operator <<' has too few parameters  
c:\program files\autodesk\maya8.5\include\maya\mstatus.h(161) : error  
C2059: syntax error : ','  
c:\program files\autodesk\maya8.5\include\maya\mstatus.h(161) : error  
C2059: syntax error : ')'  
c:\program files\autodesk\maya8.5\include\maya\mstatus.h(161) : error  
C2238: unexpected token(s) preceding ';'
...
...
Generating Code...
Creating browse information file...
Microsoft Browse Information Maintenance Utility Version 8.00.50727
Copyright (C) Microsoft Corporation. All rights reserved.
BSCMAKE: error BK1506 : cannot open file
'.\Debug\BasicLocator.sbr': No such file or directory
```

These errors were resolved when the Maya Plug-In wizard was correctly debugged. The problem consisted on a bad variable placed in the vcWizard file shipped with the application, more information can be found on the net about this subject.

## **B. Current Version**

Maya's current software version was updated through the progress of this thesis from Maya 8.5 to Maya 2008. Because the tools created for this thesis were initially developed in 8.5 it was impossible to switch to the newer version. Switching to the updated version of Maya would result in the loss of weeks of programming if the Maya base libraries were changed in any which way.

As of March 12, 2008 the current version of all applications and tools developed for this project are at 1.5. Further development on these tools is not guaranteed. To check if updated versions of the tools have been developed please check the researcher's website at [www.juanborrero.com](http://www.juanborrero.com).

### **C. Software Limitations**

There are certain limitations observed throughout the process of these tools that might be interesting to enhance later on for the Maya workflow. For example, a more precise skin cluster math or algorithm will ensure more natural organic deformations. A good example of this is the double quaternion theory plug-in developed by other programmers like Juan Borrero. The next software limitation observed in this tool was that of the IK solver math in Maya including but not limiting to the Full Body IK solution shipped with the application since Maya 7.0. The Full Body IK solution can and should be reworked to actually provide a viable solution for those seeking to drive mounted type characters without having of research heavy complex setups.

### **D. Pipeline Workflows**

#### **Rigging Basics**

Before the details of muscle setups are addressed and the manner in which the researcher came to his results, one must understand the basics of character rigging. First, one must learn the steps for creating a biped humanoid character in Maya, and the basic terminology of Maya tools.

These basic terms suggest the following questions:

**Target Audience:** Is the rig for a feature film which has no limitations, or is it for a video game with limited resources?

**Product Outcome:** Does the rig need advanced mechanics like bendy arms or stretchy joints?

**Animator's Need:** Does the rig meet the needs of the animators that requested it? Is it user friendly?

**Topology Count:** How heavy are the models in geometry? Do they have enough geometry for satisfactory deformation?

**Edge Looping:** Does the edge looping on the characters follow the contours of natural anatomy?

**System Restrictions:** Are there known hardware limitations? Is the joint count dependent of the user feedback?

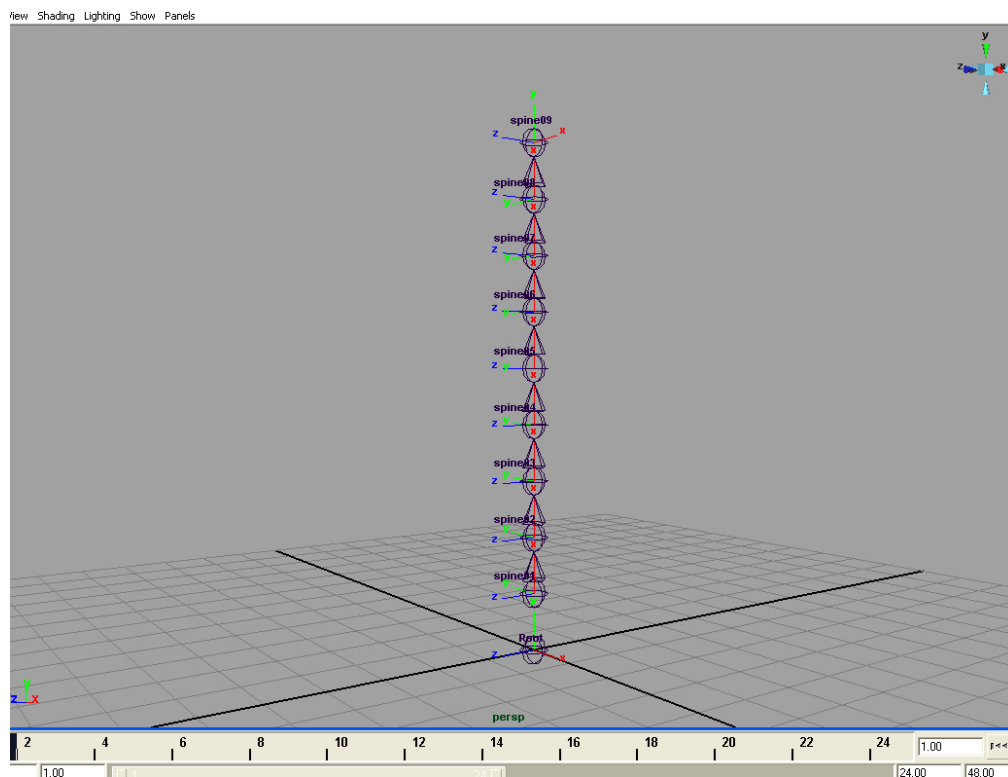
**Final Resources:** What is the final exporter maximum support? Does the system support Maya or other software nodes?

### **Spine Setup**

The spine in a biped character is composed of very basic deformation elements. Eventually, these elements will get increasingly complex, but the overall knowledge will remain the same. Standard setups run four joints as a minimum, but it could get as extensive as hundred or more joints. In this thesis project we will be running nine joints and one extra as a root. At this point, the root will not be the concern, and the focus will be

the spine joints. It is very important to notice that the orientation of the joints is correct or we could end up with a very messy non-functional rig.

**Joint Orientation:** Spine joints are very special in their orientation behavior. First off we want the Z-axis to be aiming forward rather than back. In other words Z has to be rotating on the positive axis rather than the negative. Also the Y-axis has to be aiming down the chain towards the children joints. With this in mind the orientation of the X-axis comes as a bonus because you only need two correctly aimed axis's for the orientation to be right unless we come across a Gimbal Lock.

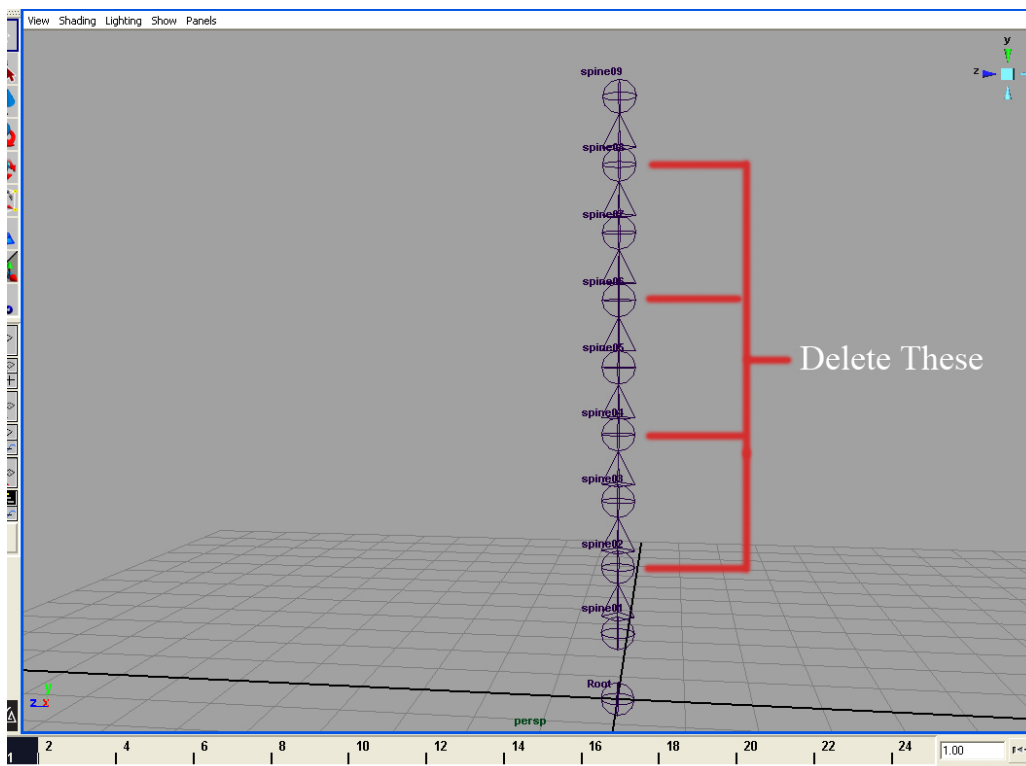


Graphic 1. Proper Joint Orientation

Now that we have some correctly oriented spine joints we can start with a simple yet advanced setup. In this setup we will have five different spines.

These spines are IK spine, FK spine, Rig spine, Twist spine, and finally the Skin spine. When all these spines are combined the possibilities for deformation are almost endless. Mixing this spine with a stretchy setup would prove very handy when animating both cartoony and realistic characters. Before we begin we should duplicate our current spine four more times so we can work with joints that are properly oriented. Let's begin with the IK spine.

**IK Spine:** First off we have too many joints for an IK spine so let's start by un-parenting all the joints and deleting every other one. (e.g. delete the even number joints, 2, 4, 6, and 8). By doing this we ensure that we have a simple five joint IK spine setup.

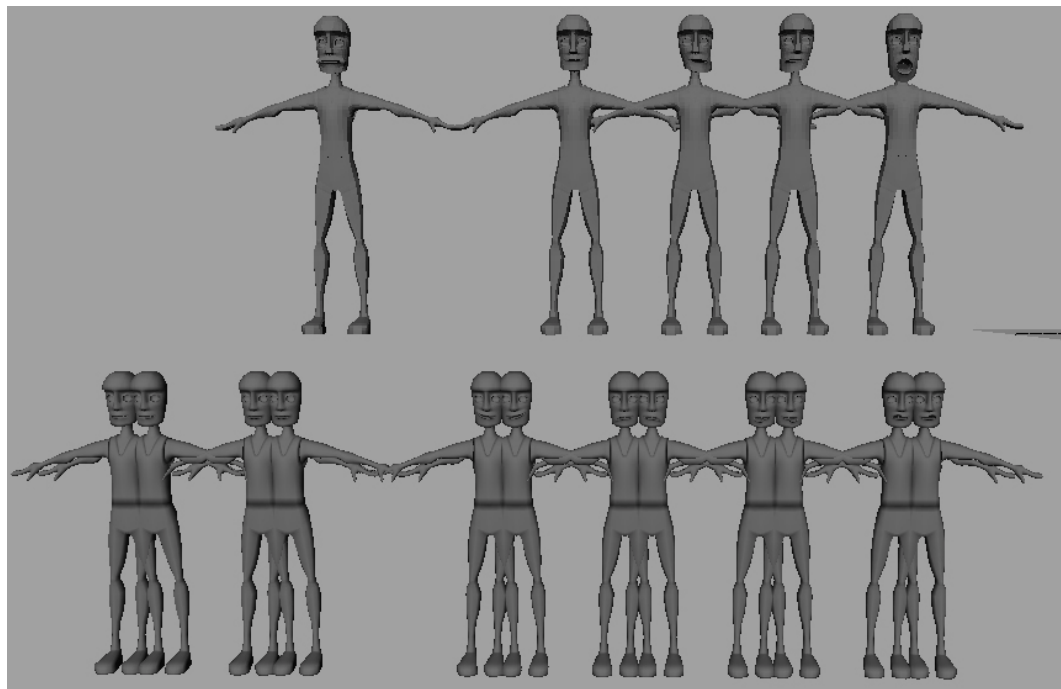


Graphic 2. IK joints to delete. Delete only after un-parenting.

With these joints deleted lets create some controls for the joints. Make sure the controls are oriented to their joints; they have zero values in all the fields, and have a rotation fix group above each of them.

### **Blendshapes**

A blendshape is a node in Maya or most other 3D packages that allows the user to change the shape of a determined polygon. One common use of a blendshape node in production is their use in facial animation. A modeler creates all the shapes of a character's expression in different heads and the rigger applies them as blendshapes to the original head.



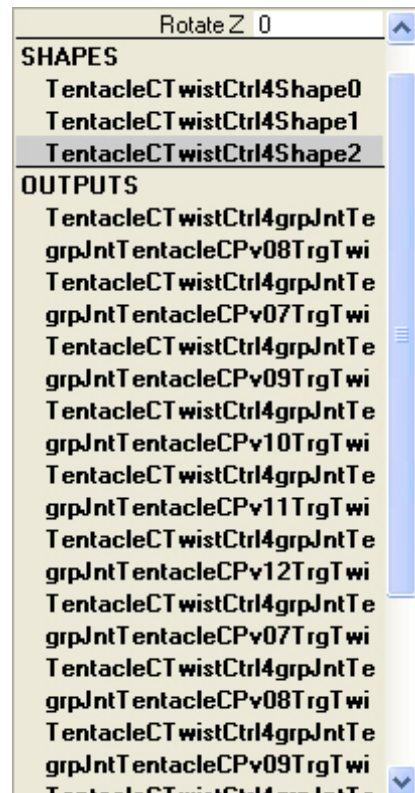
Blendshapes from Chivalry Character by Erin Lynn.

It has always been a debate between using blendshapes or joints for facial deformations. The two major arguments are that joints tend to result in loss of volume and that blendshapes are limiting. Even though they are both good in their medium an integration of the two is a sensible solution.

Blendshapes are not just used for facials; there are many other uses for this versatile tool. One can say that a blendshape is best represented when a character changes shape (e.g. Human to Werewolf). Another use is that of an influence object, like a muscle that bulges under the skin.

### Keeping History

Creation History, a riggers nightmare. History on a 3D node takes up computer resources. Many modelers hand over characters to the riggers with a history stack so immense that the scene byte size seems ludicrous. To prevent this it is recommended that one keep a reminder of cleaning history before the rigging process.



Rigging History Stack

In itself, the creation of a rig makes a extensive history stack. For this reason, it is preferred to have a clean character creation history. It is

imperative to know that after a rig is done, the history does not get erased. If the history stack gets erased after creation of a rig, the rig might stop working properly.

Moreover, there are certain history stacks in rigging that must follow a specific order. An example of this is the blendshape node. A blendshape node history stack must always be under the skin cluster history stack. A blendshape node above the skin cluster stack will result in blown geometry, double transformations, and broken rigs.

### **Joint Orientation**

By default, the children aim axis of placed joints is X. In the years of work as a rigger, this researcher has found that most pipelines have the Y-axis aiming toward the children. It is safe to assume that in the motion picture field as well as the video game field, the axis of choice to aim chains of joints at is the Y-axis.

The reason for having the joints aim toward the Y-axis is for aesthetic simplicity. Once the joints are aiming in the this way, it is natural for the animator to rotate the other joints. Also, keeping in mind that Maya has an innate Y-axis as a world up axis, it makes it easier for the new user to understand this method.

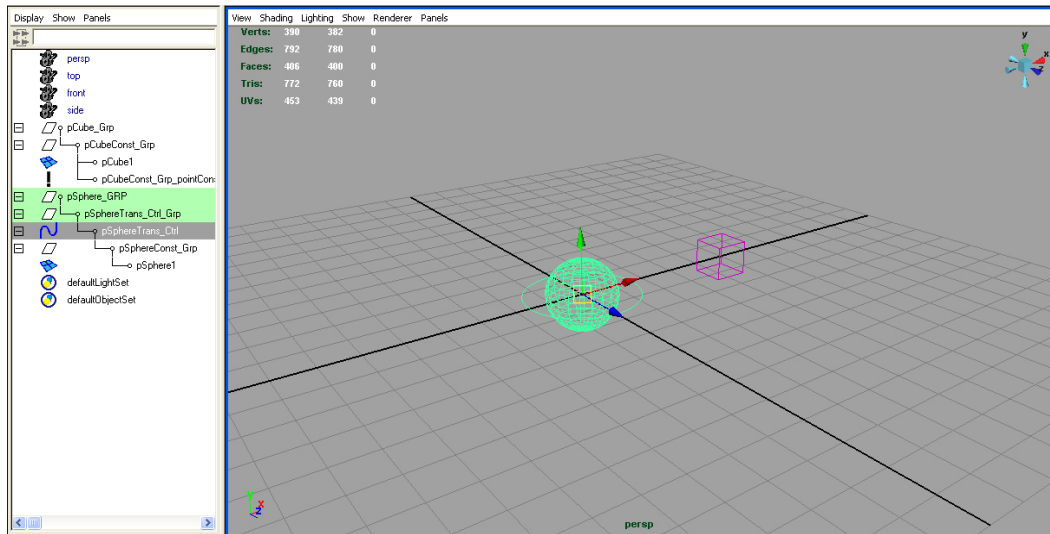
Furthermore, in Maya, only the left side of the skeleton is oriented manually or with the supplied scripts. Once one mirrors the joints from the left side, the right side of the skeleton gets oriented. This is the case if the character is symmetrical. Remember that one wants to mirror joints on

behavior not in orientation. Mirroring the joint behavior will ensure that left and right controls rotate in the same direction when selecting opposite controls. For example, when the left shoulder control rotates down, the right shoulder control rotates down as well. If a skeleton is mirrored in orientation, the animation of opposite limbs might be more time-consuming for the animator because “up” on the left control is “down” on the right control.

### **Leaving a Service Entrance**

Rigging is a very technical job and requires a lot of troubleshooting. If the rigger does not leave himself a service entrance for troubleshooting, he might find himself re-rigging a whole character pipeline. Because of this problem, it is imperative that one leaves a service entrance in order to fix a problem without re-rigging the characters. The animator must always be aware of the fact that re-rigging a character might destroy weeks of referenced animation.

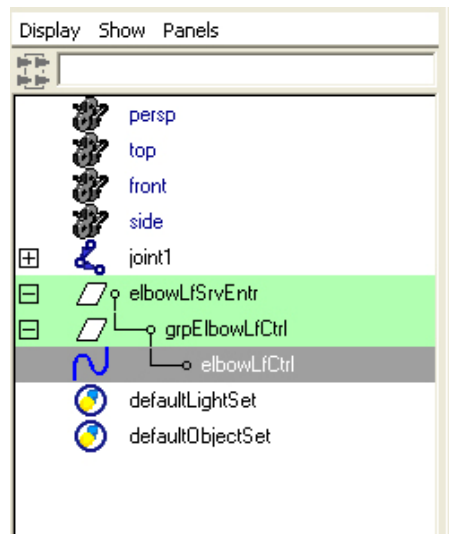
To leave a service entrance, one must have multiple groups above controls, IK's, joints, and constraints. By performing these simple tasks, one can set new driven keys, move joints and controls into new positions, and add more functionality to certain parts of the rig. All of this can be done without having to put values in the direct attributes of the selected joints, controls, IK's, or constraints. The following is a simple example of a service entrance:



## Controls Basics

Creating controls for animators can sometimes be heaven or hell. Some animators want complex controls setups while others want a minimal amount of controls that have good movement range. The type of animator with whom you are dealing is irrelevant. What remains a constant is the fact that controls need to be clean, and they must have service entrances. One easy way to have clean controls is by creating an “empty group” or “null group”, making sure it’s rotation and translation values are the same as the control it will hold. Then, one must parent the control and make sure all of its attributes are back on default 0 or 1 for scale. If the values are not 0 or 1, one must “freeze transformations” on it. Finally, the animator must create another group in the same position as the controls. This new group needs to have default values and the “null group” above the control will be parented to it. This will be the service entrance, and the animator must be sure to rename accordingly. The following is an example

of a properly created control and it's service entrance as seen through the outliner:



### **Ik's Vs. Fk's**

Many animators disagree about the way in which setups related to Forward Kinematics (FK's) and Inverse Kinematics (IK's). The concept of Forward Kinematics is based on the premise that everything proceeds in a linear path in which the parent moves first and then its children. One good example of this is an "FK arm setup". An "FK arm setup" must be animated shoulder first, then elbow, followed by the wrist, and finally the fingers. An example of an IK arm setup is when the translation and rotation of the wrist creates the translations and rotations of elbow and shoulder joints.

It is imperative that in order to have a proper IK setup, one must always have a Pole Vector control. In essence, a Pole Vector is the point at which the middle joint or elbow is aimed outward. A correctly animated Pole

Vector makes the difference between good animation and excellent animation.

### **Utilities**

Utilities are nodes shipped within the Maya software that hold different mathematical information. For example, a plus-minus-average node is a utility that has the operations of subtraction, addition, and averaging. With these types of nodes, a rig can be setup without having to write expressions or having set driven keys. Besides these benefits, utilities also provide a way of cleaning and optimizing a scene size without the fear of deleting animation curves that might be needed.

The most common utilities used on a basic rig will be reverse, plus-minus-average, multiply-divide, set range, distance between, and blender nodes. These are only a few of the many nodes inside Maya. There are also third party utilities and plug-ins that can be installed. As of Maya 8.5, there are a total of 35 utility nodes.

The correct use of utilities also allows the animator more creative freedom when creating complex animation. This is due to the fact that utilities are less limiting than expressions or set driven keys. Although their performance is better, a fully developed rig will most likely have utilities, set driven keys, and expressions working together.

### **GOD complex**

Riggers are like the life source of the animation world. Without riggers there is no animation, and without animation there is no film or short. To a

rigger, a Maya scene is a character's world, and as in real life, characters need a god, or are within something that is bigger than them. In effect this is the GOD node. The GOD node is an empty group that holds all the assets of a character. These assets include the skeleton, the geometry, the controls, IK handles, constraints, accessories, and all other elements for that character.

Under the GOD node, most of the time, one will see three other groups. These groups are named RIG, WORLD, and GEO. In the RIG group, one usually places things like IK handles, IK spines, IK curves, among other unused or un-constrained objects that cannot be scaled with the rest of the character. The RIG group usually has an unchecked Inherit Transforms option. This means that the group will remain in the same place and scale, although its parent might move and change scale.

The GEO node is another group that has all the bound, constrained, and rigged geometry of the characters. One can also find the original blendshape targets in this group if they have not been deleted. There is nothing special about this group in terms of changing its default attributes. It is simply a place where the animator can find all of the scene geometry belonging to one character.

Lastly, the WORLD node is a group that holds all rigged assets. In this group, one can find all the skeletons for a character, the controls, and constraints. This group serves as the rigger's main source of work. Within this group lie all the secrets of his rig and all his knowledge.

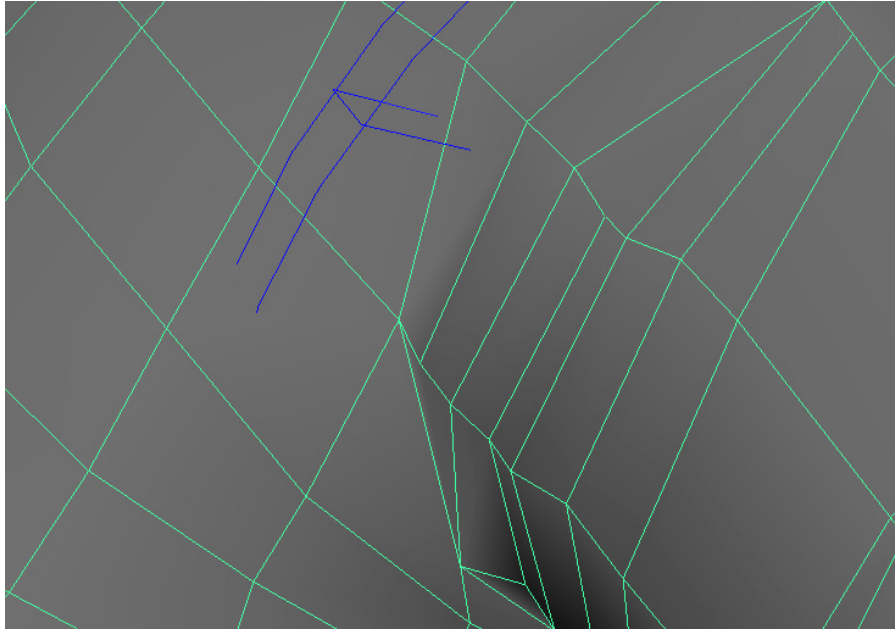
## VIII. Tool Explanation and User Guide

### A. Main Purpose

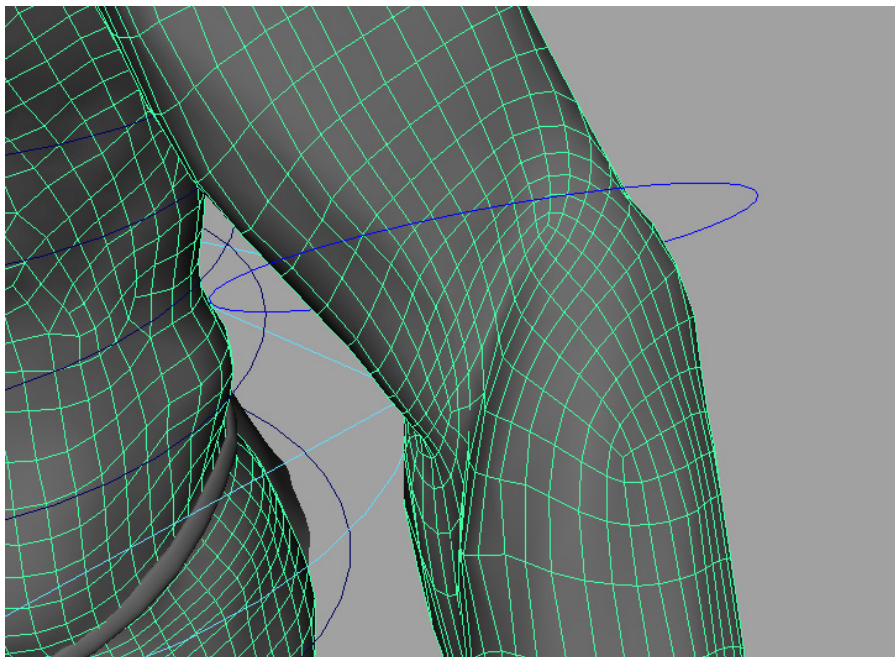
The main outcome for the creation of the tools explained in this paper changed drastically from the initial pitch to the final product. Although this researcher was disillusioned with Autodesk's purchase of and integration of Michael Comet's *cMuscleSystem* into their application Maya 2008, it helped Juan develop and think of new tools that are in demand for the application software of his use.

### B. Geometry

The layout of polygonal faces on organic type characters is very important when rigging or applying deformers. Common production mistakes in pipelines include five sided faces, five pointed stars in difficult deformation areas (shoulders, wrist, neck, mouth, hips, and ankles), excess geometry, insufficient edge looping, and lastly triangular faces in quadrangular meshes. Some of these common found mistakes are not easily overcome, but there are known fixes for them. For this reason it is imperative for Character TD's to keep a healthy informative communication environment with the modelers. The following are pictures of these mistakes in different polygonal meshes:



Five Pointed Star.



Excess Geometry

A lack of a collection of bad work was not really the purpose of this thesis. Because of this, it became difficult to find other examples for

the scenarios illustrated here, so readers are encouraged to keep an eye out for the aforementioned common mistakes.

### C. Deformers

There are four main tools developed for this thesis. Although the thesis product was named *JBMuscleSkin*, the muscle tool has been discarded. Moreover, the actual muscle interactivity has been developed and is in the tool itself and it will not be used. The next three tools created are a collision deformer, a tool to setup Maya influence objects, and a custom jiggle deformer that at the completion of this thesis is still in developmental stages.

The collision deformer has been aptly named *jbCollisionDeformer*. This deformer basically creates a squish effect on the affected vertices of a character. During the process of creating the original intent of this thesis it was discovered that a fast loading collision deformer would be a nice addition to the Maya interface. The following is a section of the collision deformer code from version 1.2:

```
aEndScale = nAttr.create( "endScale", "esc", MFnNumericData::kDouble, 3.0 );
nAttr.setReadable( true );
nAttr.setKeyable( true );
nAttr.setWritable( true );
nAttr.setStorable( true );
nAttr.setMin( 0 );

aSpread = nAttr.create( "spread", "sprd", MFnNumericData::kDouble, 0.0 );
nAttr.setReadable( true );
nAttr.setKeyable( true );
nAttr.setWritable( true );
nAttr.setStorable( true );
nAttr.setMin( 0 );

aBuldge = nAttr.create( "buldge", "bldg", MFnNumericData::kDouble, 0.0 );
nAttr.setReadable( true );
nAttr.setKeyable( true );
nAttr.setWritable( true );
```

```

nAttr.setStorable(true);
nAttr.setMin(0) ;

aHeight = nAttr.create( "height", "heit", MFnNumericData::kDouble, 0.0 );
nAttr.setReadable(true);
  nAttr.setKeyable(true);
nAttr.setWritable(true);
nAttr.setStorable(true);
nAttr.setMin(0) ;

aCollisionType = eAttr.create( "collisionType", "colTyp", 0 );
eAttr.setReadable(true);
  eAttr.setKeyable(true);
eAttr.setWritable(true);
eAttr.setStorable(true);
  eAttr.addField("planar", 0) ;
eAttr.addField("circular", 1) ;

collisionData = cAttr.create( "collisionData", "colData" ) ;
  cAttr.setArray(true);
  cAttr.setKeyable(true);
  cAttr.setReadable(true);
  cAttr.setStorable(true);
cAttr.setIndexMatters(true) ;
cAttr.addChild( aCollisionMatrix ) ;
  cAttr.addChild( aCollisionType ) ;
cAttr.addChild( aStartScale ) ;
cAttr.addChild( aEndScale ) ;
  cAttr.addChild( aHeight ) ;
cAttr.addChild( aSpread ) ;
cAttr.addChild( aBuldge ) ;

addAttribute( collisionData );

attributeAffects( collisionData, outputGeom );

return MStatus::kSuccess;
}

// -----
-
MStatus jbCollisionDeformer::deform( MDataBlock& data, MItGeometry& iter,
const MMatrix& matWorld, unsigned int multiIndex)
{
  MStatus stat;

  MObject thisNode = thisMObject();

  MMatrix invmatWorld = matWorld.inverse() ;

  MDataHandle envData = data.inputValue(envelope, &stat);
float env = envData.asFloat();

  MPlug plugGD( thisNode, collisionData ) ;
  int nCollisions = plugGD.numConnectedElements( &stat ) ;

```

```

collisionDataClass *cDataClass = new collisionDataClass [ nCollisions ]
;
MArrayDataHandle hCollisionDataArray = data.inputArrayValue (
collisionData );

int k, i ;
if(nCollisions > 0){
    for (k=0; k < nCollisions; ++k){
        if (k > 0){
            stat = hCollisionDataArray.next() ; // advance to
next one!
            if (stat != MS::kSuccess){
                cerr << "Can't next() for element " << k << "
for collisionData array. " << endl ;
                delete [] cDataClass ;
                cDataClass = NULL ;
                return MS::kUnknownParameter;
            }
        }

        MDataHandle hCollisionData =
hCollisionDataArray.inputValue( &stat ) ;

        MDataHandle hCollisionMatrix = hCollisionData.child(
aCollisionMatrix ) ;
        MDataHandle hStartScale = hCollisionData.child( aStartScale
) ;
        MDataHandle hEndScale = hCollisionData.child( aEndScale ) ;
        MDataHandle hSpread = hCollisionData.child( aSpread ) ;
        MDataHandle hBuldge = hCollisionData.child( aBuldge ) ;
        MDataHandle hHeight = hCollisionData.child( aHeight ) ;
        MDataHandle hCollisionType = hCollisionData.child(
aCollisionType ) ;

        cDataClass[k].collisionWorldMatrix =
hCollisionMatrix.asMatrix() ;
        cDataClass[k].startScale = hStartScale.asDouble() ;
        cDataClass[k].endScale = hEndScale.asDouble() ;
        cDataClass[k].spread = hSpread.asDouble() ;
        cDataClass[k].buldge = hBuldge.asDouble() ;
        cDataClass[k].height = hHeight.asDouble() ;
        cDataClass[k].collisionType = hCollisionType.asShort() ;
    }
}

for ( ; !iter.isDone(); iter.next()) {
    float weight = weightValue(data, multiIndex, iter.index());
    MPoint ptGeo = iter.position();
    MPoint ptWorld = ptGeo * matWorld ;
        MPoint ptDef = ptWorld;

```

This pretty extensive block of code covers only the creation of some attributes for the collision node and some point iterators. Point iterators are basically an array of information regarding the point in world space

of the vertices of the base mesh. The iteration process is covered by the `iter` command driven from Maya's *MPxGeometryIterator* library.

## IX. Conclusion

### A. Conclusion

#### i. Limitations

Due to time constraints the tools created in this project are at a simplistic, yet advanced level of development. It is in this researcher's belief that the tools developed in this thesis project can be enhanced and improved on to provide better user control. For example, a weights painting tool would be a nice addition to the set of tools developed with this paper. Another important tool for the Maya workflow would be a weights averaging tool based on adjacent vertices to the problematic areas.

#### ii. Problems

As with any project done over the course of a year or other time frames, it is expected to have different problems surface when creating tools for the user that is not necessarily the developer. One of such problems was the learning and understanding of the C++ object-oriented programming and Maya's API classes. The tools created with these two workflows, although catered to riggers are at some degree too advanced for certain beginner users. It is recommended

although, to look through this project and learn more in depth the width and scope of the field of animation from a programmers point of view.

iii. Results

The outcome of this project helped develop the researcher intellectually, and helped him understand the Maya API programming interface. It is safe to say that the outcome of this project was satisfactory for the individuals involved in the creation and advancement of these rigging tools.

It is in the best interest of Juan Borrero, for anyone reading this thesis to try out the tools developed at this time. Take into consideration that these tools were created for a Maya 8.5 working environment and might not work in newer versions of the software.

B. Appendix

1. Glossary

**3D** – The term 3D is an abbreviation of the word 3-Dimension, or 3-Dimensional. This is to say that when something is 3-D it has volume, weight, and perspective.

**Aim Axis** – This term refers to the direction an object points towards to either in a constraint environment or a child-parent relationship.

**Animating** – This is the process in which an artist, be it 2D or 3D, gives life like movement and characteristics to a puppet or object.

“Animation is the process of developing the actions (poses, timing, and motion) of objects.” (Maya Documentation Server)

**API** – Also known as “Application Programmer Interface” is a series of codes and libraries that are part of the used software. With API one can expand the functionalities of said software. Maya’s API is based on C++ code and is tightly integrated with MEL and Python scripting.

“The Maya API is a C++ API that provides internal access to Maya and is available on the following platforms: Microsoft® Windows®, Linux®, and Apple® Mac OS® X. You can use the API to implement two types of code resources: *plug-ins* which extend the functionality of Maya, or *stand-alones* such as console applications which can access and manipulate a Maya model.” (Maya Documentation Server)

**Arrays** – Arrays are blocks of information pertaining to multiple objects. These objects are self exclusive and they can have no relation with each other. There are three types of arrays in Maya and are as follow:

**String** – String arrays are a series of names of nodes or objects inside Maya. An example of a string array would be naming three guys {John, Karl, Mike}. When string arrays are specified, one must use the curly brackets like above. When the string is a command one can use something similar to the following:

```
String $sel[ ] = `ls -sl -typ "joint";
```

This line will save into a block of memory all the selected items of type joint.

**Vector** – A vector array is an array that holds compound information for worlds pace positioning or values. For example, color attributes like RGB are vector arrays of three decimal point values. The following is an example of a vector array used in expressions: <<1.0, 2.7, 0.2>><<2.3, 5.0, 1.0>>

**Float** – A float array is any array that has information regarding numerical values with decimal points. Although a float array can be also be used in a string array, a float array can never become a string type array because it can only hold numbers.

**Attributes/Properties** – Attributes are the inputs of an object in 3D space. The most common attributes are translate, rotate, scale, and visibility. The user can create

custom attributes in all Maya objects. All objects in Maya have keyable, and non-keyable attributes. Users can also lock, hide, and create non-keyable attributes.

“An attribute is a slot in a node that can hold a value or a connection to another node. Attributes control how a node works. For example, a transform node has attributes for the amount of rotation in X, Y, and Z. You can set attributes to control practically every aspect of your animation.

Attributes are similar to variables: they hold some value of a certain data type. The difference is that usually, setting the value of an attribute will cause some aspect of the scene to be recomputed. For example, changing the rotateX attribute of a transform node rotates its object.” (Maya Documentation Server)

**Keyable** – Attributes that can be seen and altered through the attribute editor window.

**Translate** – Attribute that controls the position of objects in the scene.

**Rotate** – Attribute that controls the rotation of objects in the scene.

**Scale** – Attributes that control the scalability of objects in a scene.

**Visibility** – Attributes that control the drawing of objects within the scene.

**Non-Keyable** – These are attributes that are not seen in the attribute editor window, but can be accessed through the connection editor. These attributes affect the matrices of an object, the pivots, and blind data of an object.

**Axis** – An axis is the point in which an objects source of rotation, translation, and scaling comes from.

**Blendshapes** – Blendshapes or Morph targets are copies of a base geometry that has been modified so that a key can be applied to change the look of the base geometry. The most common use of a blendshape is to create facial expressions on a 3D character. Although not limited to just facial expressions, blendshapes can also serve as perfect morphing tools from one character to another. e.g. A human being transforming into a werewolf. Furthermore, the Maya documentation server defines it as follow:

*“A blend shape deformer is ideal for facial animation, where you need a number of facial positions to be readily available for use in an animation sequence. With a blend shape deformer, you can set up a*

*character's face to blend between a smile, frown, smirk, and so on."*

**C++** - C++ is the name of a programming language developed in the early 80's by Bjarne Stroustrup. It was based off the C programming language and offered compatability with that and other languages available. (www.cplusplus.com)

**Character Sets** – Character sets are a grouping of all the attributes of all the controls that a character has. These sets are made to make the work of an animator easier. With character sets it is possible to separate upper body movements from lower body and even go deeper into separating hand movements among others. This in return helps the animator focus on one section of animation at a time.

**Clusters** – Clusters are in essence a locator with a connection to a selection of which it controls its translation, rotation, and scalability. Cluster are mostly used in the IK spline setups for spine deformation but have a draw back in which when controlling only one component (e.g a CV) it lacks rotation capabilities.

**Collisions** – 3D collisions are the effect generated by the computer when a surface gets in contact with another one.

Collision can be dynamic or OpenGL driven. There is little control over dynamic collision effects in Maya. There are specialized applications that deal with collisions like *RealFlow*, and the *Blastcode* Plugin.

**Color per Vertex** – Color per vertex or CPV for short is the ability of software to assign different colors to a polygonal mesh. These colors are applied on a component basis. The components usually affected are the vertices that compose the polygonal object. CPV is mostly used as a means to create visual user feedback. The colors can represent influences that other objects are exerting onto the vertices.

**Constraining** – This is the process of binding an objects attributes to the changes in another object's attribute.

**Constraints** – These are commands in Maya that bestow the attributes of an object to another. Common constraint types are point and orient constraints.

**Aim** – This constraint points an objects axis of preference towards another objects position.

“An aim constraint constrains the rotation values of an object to the translation of another object by establishing an aim vector in the options.” (Cabrera, 233)

**Parent** – Constraint that bestows orientation and translation control over another object without moving it from the objects current position.

“A parent constraint constrains both the translation and rotation values of an object to another object. The rotational axis for a parent constraint is based on the world axis, not the local axis as with the orient constraint.”  
(Cabrera, 232)

**Orient** – A constraint that controls the rotation of another object.

“An orient constraint constrains the rotation values of an object to another object. Once the constraint is applied, the rotation values of the second object will match the current rotation values of the first object.” (Cabrera, 232)

**Point** – A point constraint bestows control of the translation of an object to another.

“A point constraint constrains the translation values of an object to another object. Once the constraint is applied, the pivot of the second

object will move to the exact position as the pivot of the first object.” (Cabrera, 232)

**Pole Vector** – This is a constraint for RP type IK handles. This constraint aims the middle joint of the RP chain towards the position of the object used to constraint the IK handle.

“A pole vector constraint constrains the pole vector values of an RP IK solver to the translation position of another object or controller. Once the constraint has been applied, when the controller object is translated, the joint chain through which the IK is running through will rotate.” (Cabrera, 233)

**Bi-Directional** – A bi-directional constraint is a constraint between two objects in which both objects have control over their own attributes, but they still affect each other no matter the selection.

**Scale** – Scale constraints, control the scalability of an object to another objects scale values.

**Geometry** – A geometry constraint, ensures the translations and rotations of an object to

the boundaries of another objects geometry bounding box.

“A geometry constraint constrains the translation position of an object to a NURBS surface, curve, or polygonal surface. It does not lock the translation values of the constrained object, as other constrains do, so this allows the ability to add a point constrain to the object to another.” (Cabrera, 233)

**Tangent** – A tangent constraint ensures the translation and rotation of an object in accordance to a curve’s tangent.

**Normal** – This constraint ensures that an object translates along the normals of another object.

“A normal constraint constrains the rotation, or orientation, of an object to a NURBS or polygonal surface” (Cabrera, 233)

**Controls** – Controls are usually nurbs curves or geometry that has constraints on joints or other animatable objects. Controls ensure default pose creation through the use of default values.

**Compile** – Action taken by a programming software to create a dynamically linked library (.dll or .ml). Visual Studio 2005 is a microsoft software used to compile pages of code into a file that holds all the aspects of the program.

**CV's** – Also known as control vertices. CV's is in reality an abbreviation of these two words. CV's is the 3D term used to refer to the points in a tangent that calculate the curvature of a spline. CV's are mostly used in NURBS type geometry as they are the controllers of the surface.

**Deformers** – Deformers are nodes applied to a polygonal mesh that affect its shape. A deformer can move parts of the component level of a polygon without destroying its natural ID.

**Bend** – The bend deformer creates a change in shape of the polygonal object in the form of a bend along a common axis.

**Flare** – This deformer creates a rhomboid type shape when the polygons affected pass through its volume.

**Twist** – The twist deformer creates a helix type deformation around the polygonal meshes affected. With this deformer one can create a drill bit type of geometry.

**Wave** – This deformer creates a wave-like motion on the affected surfaces. A simple flag floating in the wind can be achieved with this deformer

**Sine** – This deformer creates a motion of mathematical proportions of the sine curve.

**Squash** – The squash deformer allows the user to flatten the volume of a surface and have it expand upon its base. This deformer is the best example to a rubber ball coming into contact with the floor.

**Stretch** – Lastly, the stretch deformer allows the user to pull opposite ends of a mesh and have extend outside its volume boundaries. Although effective this deformer does not maintain a realistic volume preservation.

**Dependency Graph** – A dependency graph in Maya is the object oriented way in which Maya interacts with its nodes. The dependency graph ensures the interconnectivity of nodes between each other. The dependency graph holds the shapes, transforms, inputs, and outputs of all the nodes and checks for loops in the connections.

“The dependency graph is one of two ways Maya represents your scene (the other being the scene hierarchy). It’s a chain of nodes.

The dependency graph is like a series of instructions for how to get the current scene starting from scratch: “create a sphere A, move these CVs, create a curve B, project curve B onto sphere A to create curve-on-surface C, trim sphere A using curve on surface C”, and so on.

The dependency graph gets its name from the connections between nodes. In the example above, the project curve operation *depends* on two inputs: sphere A and curve B.

Each node in the dependency graph represents an action to build up or change the scene, with the final result being the scene in its current state.

What this lets you do is modify or reshape input objects, change attributes on a node, change node connections, or delete nodes, and have Maya automatically and instantaneously update the entire scene to reflect the changes.

The connections between creation and editing nodes is also called *construction history*, because it records the history of how the scene was constructed.

You can view and edit the dependency graph in the Hypergraph.

You can organize nodes together using *container nodes*. Container nodes are a special type of node that lets you organize nodes into logical groupings for a special purpose. They can be used to simplify the view of dependency graph.” (Maya Documentation Server)

**Expressions** – Expressions are mathematical formulas that create multiple scenarios of behavior between objects.

Expressions are mostly used in particle dynamics to create erratic behaviors.

**Edges** – One of the components that make up a polygonal face. Edges are the lines that connect vertices together and ensure the cage of the polygonal face. It is necessary to have three edges in a triangular shape to have a polygonal face.

**Flex** – Flex is the ability of a muscle to bulge and extend under the skin. More like in real life when a body builder is flexing his body to show his musculature, the same effect can be achieved in 3D

**Forward Kinematics** – Forward Kinematics, or FK's for short is a term used in the rigging field to describe the type of movement achieved when joints are directly constraint to controls. These controls are usually parented from outer or small operations (i.e. fingers) towards bigger operations (i.e. Shoulder, Spine, Hips, etc.).

**Floats** – Float is the term used to determine those numerical values that have decimal points. The name float comes from the floating point between the full number and its decimal value.

**Faces** – Faces are one of the component levels of a polygonal object. A face is created when three or more vertices create an empty space between them. This space can be filled with a polygonal face, also known as a face.

**Grouping** – This is the action of putting multiple Maya objects under one node. Usually this node gets called among riggers the empty group or null group. Grouping can be performed by pressing the keyboard combination “ctrl+g” while having objects selected.

**Geometry** – This is the term used in animation for all type of 3D environmental shapes. This term is used to refer to polygons, nurbs, and subdivision surfaces.

**Hierarchies** – Maya is object oriented software that follows parent to child relationships. A hierarchy is basically a parent to child relationship or more so a master to slave relationship. Although Master-Slave is a rather obscure way of calling it, in reality it is more so like this type of relationship. For example, we have our “Master Node” and it controls the rotations of its slave through an orient constraint. Also grouping has a set of hierarchies that should be followed. A child for example cannot be parented to two different Master nodes, they can although constraint themselves to two or more Masters. To learn some more about hierarchies go over to the researcher’s website and look at some of the tutorials ([www.juanborrero.com](http://www.juanborrero.com)).

**Hypergraph** – Window in the Maya application that holds the dependency graph connections for all nodes in Maya.

This window is less user friendly than the hypershade, but it will graph all the blind data nodes not visible in the hypershade.

**Hypershade** – Window in the Maya application that holds buttons and icons for the creation of utility nodes, shading nodes, and all other nodes in Maya. Almost all the nodes in Maya can be graphed in this window. Rigging connections are easily made in this window.

**Inverse Kinematics** – Also known as IK's, is the process or setups that enable an animator to move and rotate from small processes to big ones (i.e. Wrist to Shoulder).

**Joints** – This is the name of the 3 Dimensional shapes used to create skeleton type structures into 3D meshes. These nodes hold information regarding their influence on the vertices of a mesh.

**Keying** – This is the action taken by an animator to create animation. It was a term coined in the early days of animation to refer at the key poses in an animation. There are multiple types of keying from keyframes, in-between keys, stepped keys, and others.

**Locators** – These are nodes inside Maya that require very little computer resources. Locators can be used as controls

and place holders for different objects. This thesis uses locators as muscle shapes.

**Local Orientation** – Local orientation is the euler angle axis of the rotate manipulator of a node in 3D space. The local orientation is different from the world orientation as this one can be in a different object world space. Refer to hierarchies for a more thorough explanation of master and slave relationships and object world spacing.

**Lattice** – Lattice is the name of a very useful deformer in Maya that creates a cluster type control in the shape of a box around selected vertices or meshes. A lattice shape can be used to create simple collisions, model characters, deform characters after rigging, and blendshaping.

**MEL Scripting** – This is Maya's innate scripting language. MEL stands for "Maya Embedded Language". With MEL scripting one can create multiple tools to help enhance the software. Most buttons, commands, and actions in Maya are executed through MEL scripting. The following is a definition of MEL scripting taken from the Maya Documentation Server:

"MEL™ (Maya Embedded Language) is a powerful command and scripting language that gives you direct control over Maya's features, processes, and workflow.

Maya's user interface is built using MEL™ scripts and procedures. When you select menu items or otherwise use the interface, Maya performs the operations by internally running MEL™ commands. You can type the same MEL commands directly as a quick alternative to selecting menu items or doing other actions.

Although MEL is technically a scripting language, it contains features that are worthwhile and easy to learn even if you have no programming experience.

You can create custom buttons on the shelf so you can run the scripts you have created at the click of the mouse.”

**MEL** – Refer to MEL scripting.

**NURBS** – NURBS is the name used in Maya for those surface types that are not polygonal and are based upon a curve mathematical solution. NURBS is in reality an acronym. The following is a description taken from the Maya Documentation Server:

“NURBS (Non-Uniform Rational B-splines) use a method of mathematically describing curves and surfaces that are well suited to 3D applications. NURBS are characterized by the smooth organic forms they produce.

NURBS surfaces can be quickly modeled and edited using a variety of techniques. NURBS surfaces are created using one or more NURBS curves that define the profile of the shape that you want for a surface, and then using a specific construction method to create the finished surface.

NURBS curves and surfaces have many applications and are the preferred surface type for industrial and automotive designers where smooth forms with minimal data are required to define a particular form. NURBS curves are ideal for defining a smooth motion path for an animated object. With NURBS, a surface can be modeled and then converted to a poly mesh.”

**Nodes** – Nodes are any object in Maya that holds a shape, a transform or both. Nodes are composed of blind data, user data, and display information. Maya primitives, Utilities, Shaders, and Plugins are all examples of nodes.

**Object Orientation** – Refer to Local Orientation

**Pipeline** – A pipeline is the workflow to follow in a studio or team of animators. There are third party and proprietary pipeline tools that facilitate the way a team works in a set environment. A pipeline ensures the timely creation and completion of tasks within a deadline.

**Parenting** – Parenting is the process of assigning one object to another. In essence this is a grouping type behavior in which one object resides within another. To execute a parenting operation select the parent object first and then the child object and press the keyboard combination “ctrl+p”. One can perform unparenting by simply pressing “shift+p”. Parent-child relationships are sometimes referred as master and slave behaviors.

**Plug-Ins** – These are developer created tools that enhance and improve the main application. In Maya there are many different plug-ins that are supplied by Autodesk and some that are on sale by other companies or developers.

**Polygons** – Polygons are triangular or quadrangular shapes in 3D space composed of vertices or CV's. These shapes are in reality complex mathematical equations that make a visual 3D element for the user.

**Rotation Fix/ Group Freeze** – A rotation fix or group freeze refers to the action taken so that skeletal controls have attributes of 0 when on default position. This is the case when the controls are in different world space locations that would result in attributes others than 0 or 1. An RF/GF is usually an empty group or group null above the control that holds its world coordinates so that the group can work inside its local axis'.

**Rigging** – This is a term used in animation that refers to the process of creating puppet-type controls on 3D characters so that they can be animated.

**Switches** – Switches are user created attributes that give functionality to multiple setups in one. One common used switch is the IK FK switch that changes the joints behavior from IK to FK without having two different characters.

**Set Driven Keys** – Set driven keys are animation curves that are linked to attributes and affect other objects. One can use set driven keys to create switch controls but it is not recommended.

**Springs** – Springs is a mathematical evaluation of all the points in the relative distance to the next point on a polygonal object. They are in essence rubber band type constraints between the vertices. Springs are mostly used in dynamic evaluations.

**Slide** – Effect a surface creates when one object rotates over another. This is the natural behavior of muscles under human or animal skin.

**Surfaces** – Surfaces are the conglomerate of faces that create an object in 3D space. A surface can be either organic in shape or non organic.

**Skin Cluster** – Skin cluster is a node in Maya that holds the information regarding the influence of joints on mesh vertices. The Skin Cluster node needs to be the last evaluating node in the history stack of a mesh. This is only because deformation nodes follow history and would create double transformations on the animated mesh. A skin cluster node that is on top of the history stack ensures that proper deformation is always present.

**Sets** – Sets in Maya are groups of objects or components for easy accessing later on during the workflow. The most common use of sets is the character sets that hold all the attributes of all animatable controls on a character.

**Script** – A script is an extensive block of code that evaluates and creates custom commands or procedures. Scripts can also be automated tools of repetitive tasks while doing certain type of work. For example an IK FK switch would be a nice script for a rigger to have handy.

**Strings** – This is the name given in Maya to all the attributes that are alphanumeric. This means, that all attributes that have letters and numbers in their value will be a string type attribute. For example, when naming objects into an array one would have to use the string operator to let Maya know that these are names and not attributes on an object.

**Textures** – Textures are the color attributes, the visual aspect, and value of characters geometry when rendered. Textures go hand in hand with rigging because deformations can stretch out perfectly mapped textures. There are two types of textures in Maya, procedural and image mapped. Procedural textures are all created within Maya using the Maya texture nodes. Image mapped textures require UV maps and hand painted images that create the colors, highlights, and shadows of the geometry. Image mapped textures are the standard for video game type geometry.

**Up axis** – This refers to the axis that points up on the world or on the object. Maya's up axis is by default the Y-axis. Other applications like 3D max have an up axis of Z.

**Utility Nodes** – These are nodes that hold some mathematical equations or procedures inside them that can be applied to other objects. Utility nodes can take various inputs and properly calculate an output value. The output value is sometimes created according to the function in which the utility is acting upon.

**Reverse** – The reverse node is simply a mathematical equation that will output an opposite value to its input. For example, if the input is 1 the opposite output of 1 is 0. The same can be said for 10 is opposite to -10. The most common use of this node in rigging is in switch controls.

**Multiply Divide** – This node takes two inputs and either multiplies or divides them and creates a result. This node is commonly used in stretch nodes, and averaging scenarios. Dividing 10 by 10 gives you 1 a more manageable number.

**Plus Minus Average** – This is a node that adds, subtracts, and averages the values it is given. This node takes in 3D values (Vectors) or single values (Floats, Integers). This node is mostly used in twisting custom attributes, and advanced stretch setups.

**Condition** – A condition node is a “what if” script statement inside its own file. The condition node takes multiple inputs and executes a set of mathematical behaviors according to the situation. It’s math is based on greater than, equal, less than, among other similar mathematical behaviors.

**Set Range** – This utility changes the distance or values of greater inputs into more manageable values. For example, -200 to 300 can become 0 to 1. This node is a simplified way of averaging big attributes into simple ones.

**Distance Between** – This node outputs the distance between to input objects. This node is useful when creating stretch type skeleton setups.

**Clamp** – Clamp utilities limit the information of their inputs, so that the output does not go over a certain range. For example, if a character can only stretch 20 units one can use a clamp to limit the stretch potential of a character to only 20 units.

**UV's** – These is the term used for the plotted points on a 3D object that hold file texture information. Properly mapped UV's will result in minimum to zero texture stretching when animating a deformable character. By default UV's are in the same position as CV's in world space but their position can change in the UV texture editor.

**Vertices** – These are the points that compose a mesh. Three vertices make up a simple triangular face. In deformable objects it is preferred to have quadrangular faces.

**Vectors** – Vectors are sets of values that correspond to coordinate on 3D space or that have three value inputs to determine the outcome. Examples of vectors are translate, rotate, scale, and color RGB.

**World Orientation** – This is a term used to refer an object that moves and interacts with the software's innate axis orientations. For example, when an object is slanted but its

movements are still parallel to those in the Maya interface (Y-up, Z-forward, X-side).

**Weights/Face Weights** – In rigging and deformation, weights refers to the influence an object exerts onto a mesh at the component level. When a mesh is bound to a skeleton, the vertices of this mesh get an influence to all the joints in the skeleton. The influence each joint exerts on all the vertices is gradual, and diminishes as joints and vertices move away from each other. Weights can be saved in a file format for later use. There are many scripts that take advantage of this resource.

**Weights Painting** – Weights painting is the process of editing the influence a deformer or a joint is exerting onto the vertices of a mesh. Most weights can be edited with the artisan brush tools that Maya ships with. Some weights can also be edited mathematically through the use of complex influence charts and file weight writing.

## C. Documentation – Trouble Shooting

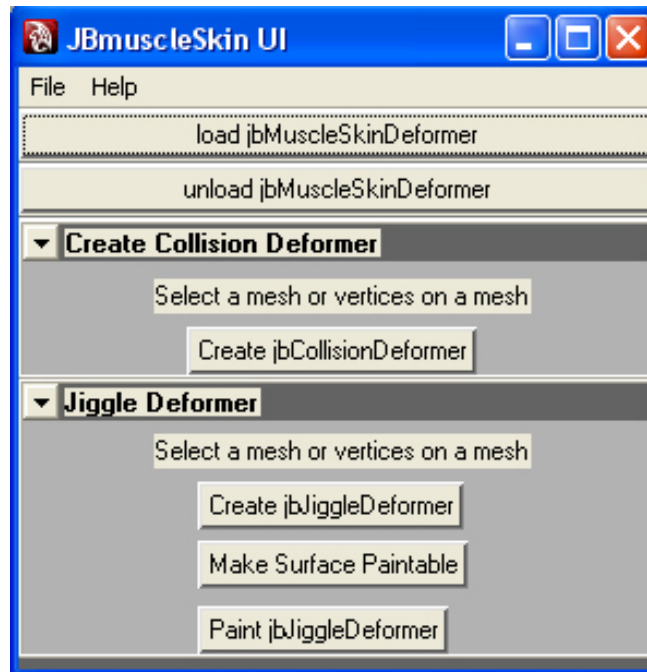
### i. Installation

1. Copy the Maya8.5 Folder to your Autodesk Maya installed directory. Replace files when prompted (default directory: "C:\Program Files\Autodesk").

2. Copy the scripts folder to your Maya 8.5 directory in My Documents. Replace files when prompted (default directory: "C:\Documents and Settings\userName\My Documents\maya\8.5").
  3. Type rehash in the Maya script editor to get Maya to recheck the scripts directory. Source the script by typing "source JBmuscleSkin.mel;" in the script editor window.
  4. Run the script by typing "JBmuscleSkin;" in the script editor window.
  5. Read the about section in the help menu for instructions on how to use the Plug-In. Remember to load the plug-in first by clicking the corresponding button in the script.
- There is an icon included in the scripts folder to use with the Plug-In, apply if desired.

ii. Interface

Interface should be self explanatory, if you are still in need of help, go to the help menu inside the tool window or visit [www.juanborrero.com](http://www.juanborrero.com). The following is an image of the final interface created in MEL script.



iii. Integration

All the tools developed are Maya 8.5 compatible and are tested to work on any working environment that uses this application version.

iv. Usage

Use these tools to create cool effects that are additions to complex rigs to enhance the capabilities of your character. The collision deformer works great for fighting characters if you want exaggerated body deformations, like when a fist hits the face. It also works perfectly for quadruped animals that when their feet come in contact with the ground they expand a little. The perfect example for this behavior is elephant feet.

v. Scripts

At completion of this thesis there was no extra scripts finished for delivery. If interested in other rigging scripts or tutorials be encouraged to head to [www.juanborrero.com](http://www.juanborrero.com) for more information.

## D. Works Cited

- i. Cabrera, Cheryl. An Essential Introduction To Maya Character Rigging. 1st. Burlington, MA: Elsevier Ltd, 2008.
- ii. Comet, Michael B. "Maya API Help Page" July 2007 – February 2008 <www.comet-cartoons.com>
- iii. Cgmuscle.com. A project by Judd Simantov and Kark Edwards. May 17, 2007 < http://www.cgmuscle.com>
- iv. Cgtoolkit.com. 2007 <http://cgtoolkit.com>
- v. Cprogramming.com. Your Resource for C and C++ Programming. August 20, 2007 <http://www.cprogramming.com>
- vi. Maya Documentation Server Version 8.5. <http://www.autodesk.com>.

## E. Works Consulted

- i. Cplusplus.com. 2008. The C++ Resources Network. < http://www.cplusplus.com/ >