

A SYSTEM FOR DESIGNING DIGITAL CREATURES BASED ON RULES OF
VERTEBRATE SKELETAL STRUCTURE

A Thesis

by

DAVID MATTHEW DRELL

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Tim McLaughlin
Committee Members,	Jeremy Wasser
	Ergun Akleman
Head of Department,	Tim McLaughlin

December 2013

Major Subject: Visualization

Copyright 2013 David Matthew Drell

ABSTRACT

Concept Designers are often required to create digital creatures that do not actually exist in real-life. These fantasy creatures are often inspired by animals that do exist, combining component body parts to create new, chimera-like forms. While these forms can look believable in stationary positions, their construction may yield awkward looking performances while in motion. This awkwardness can often be attributed to the different body parts not being connected correctly, making it impossible for the creature to be articulated in a believable way. This paper defines a set of rules, guided by study in comparative anatomy, for achieving more believable connections of body parts. This paper then details the process by which these rules are automated through a Maya script, allowing them to be integrated into a more artistic creature design process. In conclusion, it is found that the defined rules are successful in guiding believable connections. However, the implementation of the automated solution requires additional work to be a useful tool in the creature design process.

NOMENCLATURE

3D	Three Dimensional
CV	Control Vertex
IK	Inverse Kinematics
MEL	Maya Extensible Language
UI	User Interface

TABLE OF CONTENTS

	Page
ABSTRACT	ii
NOMENCLATURE	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES	vi
1. INTRODUCTION	1
2. RELATED WORK	4
3. METHODOLOGY	8
3.1 Artistic Control	8
3.1.1 Workflow Integration	8
3.1.2 Working Environment	10
3.1.3 Extensibility	11
3.1.3.1 Modeling	12
3.1.3.2 Part Organization	13
3.1.3.3 Skin Setup	14
3.1.3.4 Bone Setup	15
3.1.4 Output	17
3.2 Anatomical Rules	19
3.2.1 Overview	19
3.2.2 Major Anatomical Part Rules	20
3.2.3 Bone Connection Rules	22
3.2.3.1 Limb Bones	22
3.2.3.2 Scapula	23
3.2.3.3 Vertebra	24
3.2.3.4 Head Bones	25
3.2.3.5 Ribs	27
3.2.3.6 Pelvis	28
3.3 Allometric Scaling	29

	Page
4. SYSTEM IMPLEMENTATION	31
4.1 Technical Specifications	31
4.2 User Interface	32
4.3 Part Connection	35
4.4 Bone Connection.....	36
4.4.1 Default Bone Deformation	36
4.4.1.1 Type A Deformation	37
4.4.1.2 Type B Deformation	41
4.4.1.3 Type C Deformation	45
4.4.1.4 Type D Deformation	49
4.4.2 Special Case Method	50
4.4.3 Allometric Scaling	53
4.5 Skin Connection and Deformation	59
4.5.1 Default Method	59
4.5.2 Additional Deformation for Added Vertebra	62
4.5.3 Alignment of Borders Between Parts	67
4.5.4 Deformation Smoothing	69
4.6 Disconnecting Parts	77
5. CONCLUSIONS	81
6. FUTURE WORK	83
REFERENCES	87
APPENDIX A	89
APPENDIX B	90

LIST OF FIGURES

		Page
Figure 1	The Chimera of Arezzo	1
Figure 2	Jar-Jar Binks w/Reference Animals	2
Figure 3	Example Part Organization for a Dog's Right Lower Leg	14
Figure 4	Major Anatomical Parts as Applied to Different Tetrapon Types	19
Figure 5	Primary (Red) and Secondary (Green) Major Anatomical Part Cuts	21
Figure 6	Standard Limb Bone Connection Steps	23
Figure 7	Standard Scapula-to-Humerus Connection Steps.....	24
Figure 8	Standard Vertebra Connection Steps	25
Figure 9	Undesired Uniform and Single Axis Skull Deformation to Connect to Vertebra.....	26
Figure 10	Standard Skull Connection Steps (Realized as a Vertebra-to-Skull Connection)	26
Figure 11	Standard Rib-to-Scapula Connection Steps	27
Figure 12	Standard Pelvis-to-Femur Connection Steps	29
Figure 13	The beastMaster Window	33
Figure 14	Transforming Parts Within the Maya 3D Window	35
Figure 15	Typical Type A Deformation Joint Layout	37
Figure 16	Limb Bone Connection by Length Scaling	38
Figure 17	Matching Orientation and Scale of Connected-to Bone	40

	Page
Figure 18 Typical Type B Deformation Joint Layout	41
Figure 19 Part Curves Generated Through Vertebra and Joined to Create a Vertebra Curve	42
Figure 20 Vertebra Placement and Orientation Along Vertebra Curve	45
Figure 21 Typical Type C Deformation Joint Structure	46
Figure 22 Determining Target Point for X-Axis Stretching	48
Figure 23 Typical Type D Deformation Joint Structure	49
Figure 24 Typical Special Case Deformation Joint Structure	51
Figure 25 Relative Positioning	52
Figure 26 Procedural Center Point Determination for a Single Vertex	54
Figure 27 Locator Placement to Define Connecting Extrusion	64
Figure 28 Reorienting Extrusion Border based on Matching Border Normals.....	65
Figure 29 Starting Point of Border Matching Based on the Closest Pair	68
Figure 30 Matching Border Positions Based on Ordered Vertex Pairs	69
Figure 31 Example Before and After Smoothing of an Uparm-to-Torso Connection	70
Figure 32 Smoothing Falloff Between Parts	71
Figure 33 Two-Step Neighboring Vertex Positions Considered for Determining Smoothing Position of Center Vertex.....	73
Figure 34 Smoothed Green Points Regain Their Volume by Ray-Casting Against the Original Surface and Finding the Point they Intersect (Represented by Orange Points)	76
Figure 35 Disconnection Types as Performed on a Deformed Leg	77

1. INTRODUCTION

Fantasy creatures have been a frequent presence in fictional works, stretching from mythology, to fantasy, to science fiction. Designing these creatures in a way that is plausible to the viewer has been a constant challenge to the artists who create them. In order to reach this level of plausibility, the artist will commonly design not solely by creative whim, but by referencing existing animals found in nature. Examples can be found in works throughout the ages, from depictions of mythological creatures such as the minotaur, the centaur, the satyr, and the sphinx, to modern film aliens, ranging from the classics of Star Wars to more recent depictions in John Carter and 8MM. All these imagined creatures spanning from the chimera of the Iliad (Figure 1) to Star Wars' Jar-Jar Binks (Figure 2), draw on the structure of existing animals to dictate their design.



Figure 1. The Chimera of Arezzo

Artists use this technique of modular design because the viewer is predisposed towards the way nature will evolve a creature, a base understanding of the way a creature must be structured to function within the physical laws of our world. [Page

accurate underlying skeletal structure that the artist could then use to embellish upon outside the system. By attaching the various parts of the creature together in a way that is consistent with the way similar animals are structured, this system guides the artist toward design choices that will enable the animation of a fantasy creature to function plausibly when in motion.

This system is implemented as a Maya script tool with a minimal user interface. By creating this system as a tool that operates within Maya, it is able to leverage Maya's polished 3D interface to provide the majority of functionality the artist will require will manipulating the design. Throughout the design experience, the system will enact an established set of anatomically-based rules to create connections between parts. This connection process, and the rules that guide them are automated and hidden from view, allowing the designer to function on a strictly creative level without concern for the underlying mechanics.

The impact of this tool lies in its ability to remove the burden of anatomical knowledge from the artist, allowing amateurs to create more professional designs and generally freeing up the mind to think creatively. The results produces a creature whose form conveys a more believable underlying anatomy, ensuring it doesn't break the viewers suspension of disbelief. It will also produce a skeletal structure for the designed creature that can guide both additional detailing in design and modeling, and also benefits the rigger in understanding and creating the creatures articulation, allowing for more plausible performances through animation.

2. RELATED WORK

The recognition that creature design instructed by existing biological structures leads to more plausible fantasy creatures has long been recognized by traditional artists. Whitlatch advises that, “by doing realistic, anatomical drawings of the muscles and skeletons of real life animals...you will gain the necessary experience needed to design original creature anatomy.” [Whitlatch 2013] Huante makes a similar point when he states that designing a creature with a believable structure is, “completely intuitive after gathering information, watching nature programs, watching people.” [Huante 2006] In the use of digital creatures for animation and visual effects an adherence to anatomical design is beneficial to the process of developing digital characters and to the plausibility of their performance. The underlying geometry affects the outlying form of the digital creature and it is important to have these alterations in form based on actual anatomy, rather than artist whim. Digital creatures all share certain properties, one of which is that, “novel or fanciful features may generally be dissected into component parts based upon realistic creatures.” [McLaughlin et al. 2007] It is this combination of component parts that this thesis seeks to facilitate. The method by which these parts would fit together in nature is not necessarily evident, especially to an artist who is not well versed in comparative anatomy.

McLaughlin states that, “for most digital creature work, form, or questions of anatomy and geometry determine the complexity of the job at hand more than any other issue.” [McLaughlin 2005] Therefore, the focus of this thesis is the development of a

system that supports the design of plausible forms for fantasy creatures by artists who have limited knowledge of anatomy. The results of this thesis will allow the artist to create a design that addresses all three of these elements by allowing the artist to rough out a form that has underlying anatomy and a rough sense of the geometry required.

For all designers, in particularly for novice artists, reference material is critical. Page states that, “its really risky when you start making up things with characters regarding skeletal systems, muscles...Nature has a way of doing this stuff correctly, so it helps if you refer to nature as a means to educate yourself, so that when you are trying to be creative what you do makes sense in terms of anatomy and the way animals are.” [Page 2004] McLaughlin further highlights the need for reference material to create plausible creatures, especially when creating a “naturalistic” [McLaughlin 2005] digital creature. The approach taken by this thesis project, however, is to replace the use of direct reference with the use of preexisting components that have themselves been generated from reference material.

This system dictates surface form based on an underlying anatomical structure. Prior work into anatomically based modeling was examined to determine how to achieve a highest level of realism. As “Anatomically Based Modeling” by Jane Wilhelms and Allen Van Gelder suggests, “in general, computer graphics has achieved greater realism by developing methods that simulate the real world, rather than using ad hoc methods.” [Scheepers et al. 1997] In keeping with this assertion, this system makes use of a realistic skeletal structure beneath the surface of each component of the designed creature to allow any changes in its anatomy as a result of the designers manipulation to

be made evident in the surface form. Schepers, Parent, Carlson, and May state in their paper, “Anatomy-Based Modeling of the Human Musculature” that, “by analyzing the relationship between exterior form and the underlying structures responsible for creating it, surface form and shape change may be understood and represented best.” [Wilhelms et al. 1997] By implementing a system of surface deformation similar to those described in these works, the system attempts to achieve equally well-represented underlying anatomy.

As this thesis will create a base form that will likely require further detailing on the part of the artist, it was important to establish that this was indeed possible given the output mesh provided by the implemented system, a reasonably high density mesh with complex topology. Given the high density of the mesh and the desire to provide the designer with as great a level of artistic freedom as possible, examination of prior work in the field of digital sculpting was essential. Marie-Paule Cani and Alexis Angelidis explore three methods of digital sculpting in, “Towards Virtual Clay”. The goal of the techniques described in this paper is to ideally allow a user to, “be able to deform, add, and remove material freely in real time, without any geometric or topological constraints on the modeled shape.” [Cani et al. 2006] With this ability and the base form output by this thesis to work off of, the designer should be able to create any sort of digital fantasy creature they can imagine that will retain anatomical accuracy (provided they do not stray too far from the original form provided).

In, “The Amateur Creator”, Stephen Boyd Davis and Magnus Moar discuss designing software for amateur users. The system produced by this thesis is targeted at

those with a limited to nonexistent knowledge of anatomy, making this work eminently relevant. In Davis and Moar's work, they define an amateur user as being different from a novice user, as amateurs are, "not necessarily in transition towards being an expert." [Davis et al. 2005] This is an important distinction, as the system will hide all underlying anatomical elements from the user, making it solely an aesthetic endeavor from the user's perspective, not attempting to help the user learn about anatomy in the least. The paper also describes five "transformations" that make a system appropriate for amateur use: foregrounding, backgrounding, automation, constraining, and integration. The system attempts to adhere to these transformations, ensuring it is appropriately for the intended skill level of the user.

Finally, prior work into other methods for designing believable digital creatures, other than that proposed in this thesis was examined. Karl Sims paper, "Evolving Virtual Creatures," and the more recent "Evolving Virtual Creatures Revisited," by Peter Krčah uses genetic algorithms and physically based simulations to generate a wide range of creatures that move in a believable fashion. However, the user has limited aesthetic control, given the systems automated method. Sims states that should greater aesthetic control be desired, the user could set fitness values by hand, but would "require too much patience on the part of the user." [Scheepers et al. 1997] Alternatively, the user could pick from a selection of generations and further evolve, offering limited control, but gives, "some influence on the creatures that are developed." [Scheepers et al. 1997] For the purposes of creature design with a mind towards a specific purpose, this level of artistic control is clearly unacceptable.

3. METHODOLOGY

3.1 Artistic Control

3.1.1 *Workflow Integration*

It is unlikely that an artist could be convinced to use the tool produced by this thesis if it required they adopt a totally new workflow in order to gain the benefits of the system. It was therefore a major goal in the development of the tool that it blend into the standard creature design workflow as cleanly as possible. In doing so, workflow disruption is kept to a minimum. In order to establish what a standard workflow was, the design processes of professional creature designers, Carlos Huante, Neville Page, Terryl Whitlatch and Puddnhead was observed. Although their exact approaches differed, there was sufficient similarities to establish a set of elements integral in the design of the tool.

The designers generally began their process by trying to get the basic form of the creature. This generally consisted of drawing simple shapes until a silhouette began to form. The artists frequently began with a circle representing the head or the body and then built off of this initial form. Given this tendency to build off the initially established part, it was deemed important to have the tool work in the same fashion. Therefore, when appending parts onto the form, the system allows existing parts to remain static, while altering only the appended part as the system requires. As such, the designers can build onto the model just as they would build onto their primitive forms.

This is implemented in the tool by ensuring that when the artist makes a connection, only the connecting part is deformed, while the connected-to part remains static.

During Huante's design of the basic form, he found his design required a major change, stating, "now that I drew this upper body, it tells me that this lower torso is just not working...and this is the process, going back and forth." [Huante 2006] This process was mirrored in the workflow of the other artists as well. Because the artists generally had no prior concept of what they were trying to achieve, the form or silhouette they were creating changed several times before they settled on something they liked enough to begin detailing. This is included in the tool in the form of a disconnection tool that allow the artist to easily disconnect a part for reorientation or replacement with a different type of part. This is supported by implementing part connections virtually, such that they appear to be mesh connections, but actually remain discrete.

The workflow of the artists also required they have the ability to be creative and unconstrained. It was therefore important to ensure that this flexibility be carried on throughout the system. While the system guides the artist in terms of the skeletal connections it allows, it is nonrestrictive in terms of enforcing other aspects of the creature that would cause it to be implausible, such as artist-applied isometric scaling, application of limbs incapable of supporting the torso, or unstable postures and stances. In allowing these artistic decisions, the system allows the artist to assert things such as a changes in gravity and other factors that could allow such a creature in a non-Earth-Normal environment.

In addition to giving the artist the freedom to stray from the bounds of what is plausible, it is important that the system grant the artist the freedom to be creative and design areas of the creature after its anatomical structure has been established. As such, it is important that the library of Anatomic Parts are not too detailed so as to lead to designs that, “look inconsistent because they look like they [the designer] put a zebra head on a tigers body on a weasels lower legs... You’ve seen people do that and it looks like it. Creativity should be pure.” [Huante 2006] In keeping with this, the supplied animals are of low enough resolution to merely provide the basic shape of the creature, allowing the artist room for embellishment.

3.1.2 Working Environment

The suggested space for this workflow in the design process is as a replacement for the early design stage, which generally occurs as the artist quickly roughs out form on paper. Therefore, design through this system must be similarly quick and easy. As the majority of the design comes through manipulation of models in a 3D space, the modeling environment provided to the artist must have a well-thought out, simple user interface. Rather than attempting to custom create such an environment, it was considered best to leverage the features of existing 3D Modeling software, building the functionality of the script on top of it through the program's available extensibility. This allowed focus to remain on developing the core ideas of the thesis, rather than spending time developing a 3D Environment and UI from scratch.

Another benefit of building upon existing software is that it allows for a sense of familiarity for users who are already familiar with the base software the script is built

upon. As the UI elements introduced by the script are minimal, familiarity with the base software essentially removes the majority of the learning curve required to design in this workflow, encouraging its adoption by the artist. In order to gain this benefit for as many users as possible, it was important to select a base software that was widely used. For the purposes of this thesis, the selected base software was Maya, being one of the main industry standards, ensuring that familiarity would be likely amongst the target users of this thesis.

While Maya was deemed the best bet in terms of widespread familiarity, and therefore used in the development of this thesis, it was also recognized that it is not the only software package in wide use for the purposes of 3D modeling. There was therefore effort put into coding this thesis in such a way as to make porting it to other software packages as simple as possible. By potentially making the script available in a number of different packages, the number of users familiar with the base software increases. In order to allow for simple porting, this thesis was created in a software agnostic scripting language, Python. In addition, while Maya functions were used to enact much of the functionality of the script, an effort was made to stick with basic functions that would be present in other software packages and custom coding many of the more complex operations.

3.1.3 Extensibility

In order to allow a community of users to extend the usefulness of this system beyond the original scope, the ability to add to the list of available animal types was a key feature. The goal of this extensibility was to create a simplified process, such that

the barrier for entry to creating these new types was very low, allowing anyone with a rudimentary understanding of the system to generate new types. In addition to some modeling restrictions placed on the user when modeling bone and skin meshes, setup has been implemented as a script tool in Maya that the user will utilize to prepare the new animal model for use. These setup tools will properly structure the parts and create all data files (Appendix B) required by the system.

Upon completion of setup, the newly created parts can be placed in the *beastMaster/animals* directory. The list of available animals in the *beastMaster* tool is generated at run time by searching through this directory, facilitating the simple installation of additional animals into the system.

3.1.3.1 Modeling

The process of setting up a new animal type begins by modelling the skin and skeleton of the creature as mm models. When creating the bone models, there are no topological restrictions on bones, allowing resolution to be as low or high as the user modeler wishes. This lack of restriction also allows for the possibility of using a high-resolution 3d scan of a skeleton. One caveat to this is that there are a number of per-vertex calculations used during the connection process, so high resolution models have the potential to dramatically slow down the deformation process.

In contrast to the bones, there are topological restrictions on the skin. These restrictions dictate how many points can be used to define a connecting border. This is because skin connections require a one-to-one vertex connection, so the vertex count for each part border must remain consistent between animals. Outside of border edges

however, there are no restrictions. This could potentially allow for a high resolution model such as a 3d scan if resolution was reduced to the proper count at the connection border. It is debatable whether a high resolution model with low resolution at connection borders would be of good visual quality so it is recommended that skin models remain of low resolution. The use of low resolution models is further recommended due to the fact that one of the primary uses for the output of the system, and one of the artistic goals is to provide a low resolution base mesh for 3d sculpting.

3.1.3.2 Part Organization

Once skin and bone are modeled, the skin must be divided into the relevant parts. This division of the skin into parts can be done with any available modeling tools to create borders with the prescribed vertex counts. Presuming the entire animal has been modeled, this will likely result in 17 separate models, although that number could be lower if the animal in question is lacking a part, such as a tail.

When the user has divided the skin into parts and created bones, parts need to be grouped into a specific format (Figure 3) so the beastMaster script knows where to locate them during the connection process. The Setup tool automates this grouping under the UI's Part Organization tab, so the user does not require specific knowledge of the structure. This Setup tool does, however, require precise naming of the parts, so the user must be familiar with the naming conventions of the system. Should parts be named incorrectly, the user will be alerted to this fact by the setup tool.

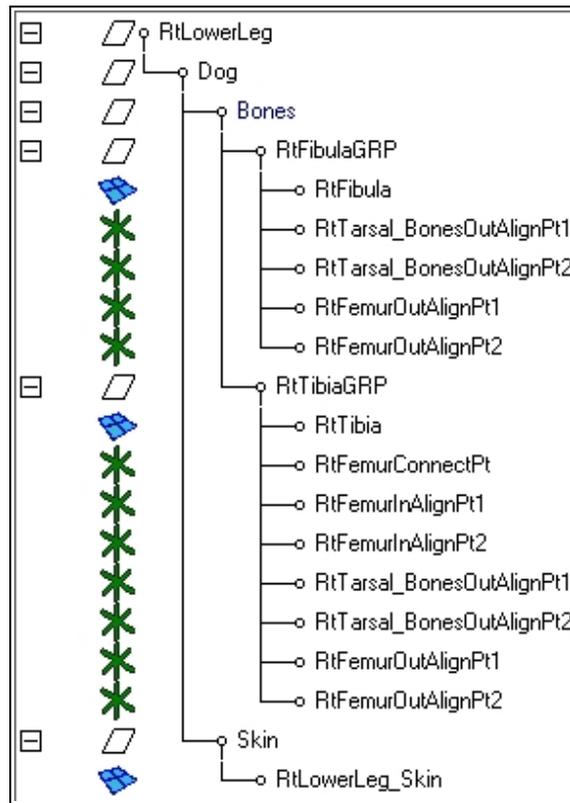


Figure 3. Example Part Organization for a Dog's Right Lower Leg

3.1.3.3 Skin Setup

With the part properly organized, each skin and bone element must be setup individually. In the case of the skin, this means saving out skin attribute files to disk. These files are generated by the Setup tool, under the UI's Skin Setup tab. In order to setup a skin object, the user must first fill the part name field, specifying the skin part that will be worked on. Once the current part is specified, the setup tool UI is populated with border edge fields, each corresponding to a part that the current part can potentially connect to. The user must then select the edges that comprise the border associated with the field and press the corresponding button to populate each field.

Once all edge fields have been filled, the Skin Attribute File can be written out

by the Setup tool. In the case of parts that have a sidedness, the user will have the option to also write a Skin Attribute File for the opposite side. Because this opposite side skin attribute generation uses the same edge indices for both sides, it is essential that left and right sides have identical topology.

3.1.3.4 Bone Setup

Finally the user must setup each bone in the part for connection. This setup process will generate the Bone Attribute file, as well as any locators required to facilitate the connection process. The user begins by specifying the bone to be worked on, as well as any bones it will connect to. For each of these bones, the Setup tool will determine the connection type this connection requires and record it in the Bone Attribute file.

The Setup UI will also add any additional fields that it requires for specific bone connections as connections are specified. Examples of additional fields include: specifying a vertebra to be copied in the case of vertebra to vertebra connections, alternate joints to be used for rib to scapula connections, and bone joint diameter for limb bone connections.

Once all bones involved in the connection process have been defined, the user must define the position of the joints that will handle bone deformation. In order to simplify the creation of these joints, placement is defined by the average position of a specified set of vertices on the bone, rather than requiring manual positioning in space by the user. The user will define this set of vertices for each joint at this time by selecting as few as two vertices in the 3D Viewer and populating the field for each required joint.

With the joint chain defined, bone vertices must be weighted appropriately. To avoid the cumbersome process of weight painting each bone, the user is only required to correspond each vertex with a single joint that will affect it. The deformation process within the beastMaster script has been setup to accommodate for the limitations of rigid binding. Where these limitations cannot be surmounted, the Setup tool will automatically generate a smooth weighting based on the user-specified rigid binding.

For simplicity, the user is only required to specify a single joint chain for each bone, rather than a chain corresponding to each bone it will connect to. In order to create a successful connection at deformation time, this joint chain will need to be reversed for some connections. Therefore, the user is required to specify whether the joint chain should be created in the order specified, or in reverse, for each possible bone connection.

Once all fields have been set, the user must test the resultant setup. The user may test against any of the available connections. Testing involves generating the specified joints and binding in the direction attributed to the current connection. This gives the user the opportunity to apply transformations to the joints to make sure bindings are as desired, as well as evaluating the position of the generated joints. At this point the user may decide to adjust any of the fields to yield a better setup or generate the Bone Attribute file and alignment locators.

Once the bone setup process has been successfully completed, the Bone Attribute file will have been written to disk. However, it is important to note that the generated locators have not been saved. These locators should be saved once all bones in the part

have been successfully setup. At this time, the user will export the individual parts into their own files by clicking on the part's top level group and exporting the selection.

The exported part is now fully setup and ready to use with the beastMaster script.

3.1.4 Output

When the user completes the design process, the provided output comes in two forms. The first is a collection of mm part models combined together to represent the skin of the designed creature. With the collection of mm part models, the artist can pursue any of the three of the proposed ways of continuing the design process at this point. These proposed methods involve embellishing on the design by printing the result out on paper to use as a silhouette for adding hand-drawn details, utilize 3D printing techniques to create a base for maquette sculpting, or as a base mesh for 3D sculpting in a software package such as zBrush or Mudbox.

For standard printing, the models are prepared. Simply choosing a desired camera angle and capturing a screenshot (or setting up a full render) will achieve the desired results. For the other two methods, which likely require a single contiguous mesh, the user must take the extra step of combining all part meshes into a single model and merging co-located vertices. This is necessary due to requirements of the system, that dictate the models remain discrete through the design process. Once completed, this model can easily be exported for use with a 3D printer or import into a 3D sculpting package. The mm format of these models was selected, in part, due to their pervasive support in these downstream processes.

The second output at the close of the design process is the underlying skeletal

structure. This is represented by a set of mm models, each representing a single bone in the skeleton. The prescribed uses for this output skeleton is twofold. The first use is to further guide embellishment of the output skin model. During the automated connection process used to guide the design to this point, the skeleton has been integral in dictating the surface form. Once this automation is no longer in play, it is advisable that the artist continue to reference the skeleton, such that further embellishment does not diverge too much from the underlying anatomy. Additionally, limitations in the deformation and smoothing methods used in this script can sometimes cause the skin lose some of its correspondence to the underlying skeleton. It is left to the artist to correct these failings at this time, guided by the output skeleton.

Beyond the design process, it is suggested that the underlying skeleton be provided to the creature rigger. To this end, the skeleton provides a useful guide towards joint placement during the articulation process. It also provides the rigger with an idea of how the creature will move, mechanically, and where additional anatomy, such as tendons and muscles would connect, dictating skin deformations.

3.2 Anatomical Rules

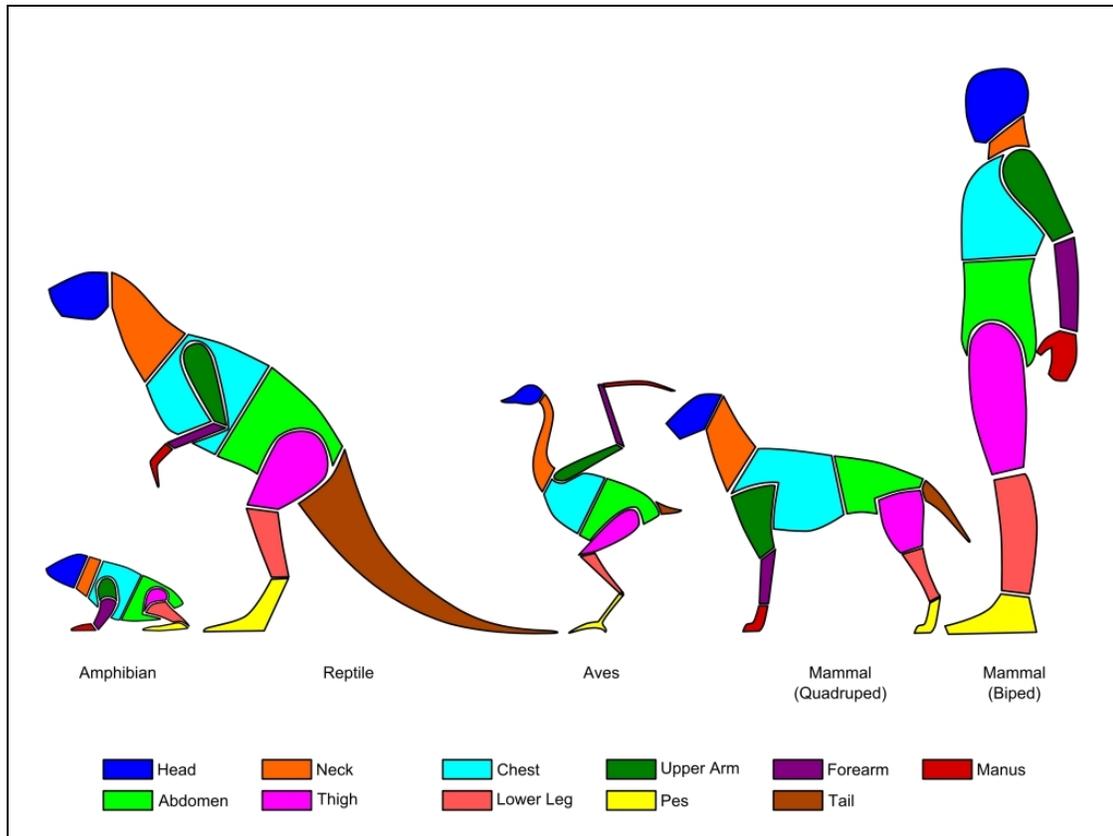


Figure 4. Major Anatomical Parts as Applied to Different Tetrapod Types

3.2.1 Overview

Before creating a system for automating the creature design process, it is necessary to establish the anatomically guided set of rules for deforming the skeletal structure. Following along the observation that, “Animals within a taxon above the family level tend to share basic structural patterns,” [Hildebrand et al. 1998] it can be noted that animals within the superclass Tetrapoda all share a largely parallel skeletal

structure. While these structures may diverge in terms of their evolved form and function, in a general sense these structures all fit together in a very consistent way, for all but a few exceptions. It is this observation that allows the creation of a set of rules that are generalized enough to allow simple extensibility within the system (Figure 4).

3.2.2 Major Anatomical Part Rules

The first rules that had to be established was how to split animals into the major anatomical parts that the designer would combine to create the resultant creature. One rule guiding the division of these parts was established that cuts had to occur along joint connections, such that a single bone would not exist in two different parts. The primary reason for this is so that parts can be attached by matching bones at their joint connection. This also serves the purpose of making it obvious where parts should be cut, given the variety of forms tetrapodal animals present. This is most notably demonstrated in defining the division between the foot and lower leg parts of animals with different standing grades, where cutting based on similar proportions would yield parts containing different sets of bones between animals.

The second rule that guided how the major anatomical parts would be divided is that parts must be were numerous enough to provide significant design options, but not so numerous as to make setup tedious to the user. This rule is more artistically guided than based on anatomical rules. Upon examining the Tetrapodal body and skeletal structure, an obvious point of division is where limbs, neck, and tail connect to the torso, due to the generally large change in shape and the obvious joint connection point.

A full limb or torso also represents a mechanical system that has evolved to

function as a whole. While maintaining these systems as a single object would help ensure a structurally sound, and by extension, believable creature, this level of division does not leave enough room for artistic freedom. As this system is attempting to strike a balance between the two, further subdivision was deemed necessary (Figure 5). Should the user wish to maintain these systems as a whole, they can easily import the subdivided limb or torso into their default position and connect them immediately.

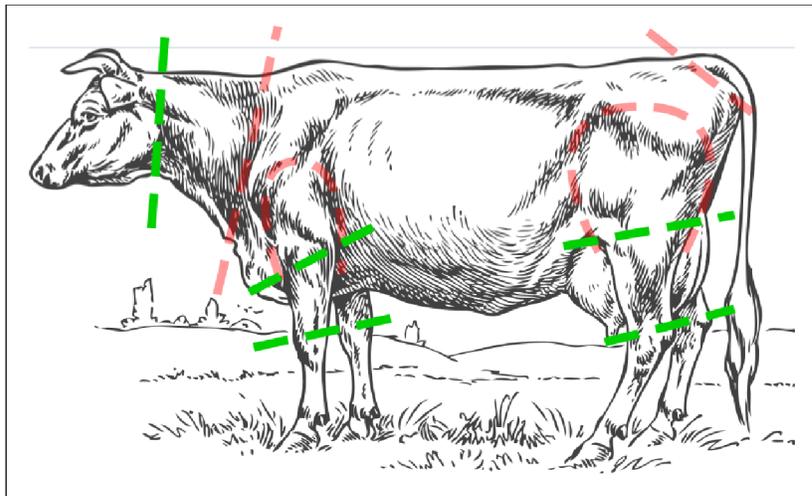


Figure 5. Primary (Red) and Secondary (Green) Major Anatomical Parts Cuts

In order to further subdivide, additional obvious shape changes are noted, found at the connection of manus and pes to limb, head to neck, and chest to abdomen. Divisions can also be made in the limbs based on the major joint connection point of the knee and elbow. Some consideration was given to creating separation at each phalange, allowing the designer to add digits from different animals to the hand/foot parts.

However, given the generalized resultant mesh, this was determined as falling into the range of tedious setup. At this level of subdivision, the designer has 17 parts per animal with which to combine, as listed in Major Anatomical Parts (table 1).

3.2.3 Bone Connection Rules

Although it is assumed that each tetrapodal animal's skeletal system will connect in the same way, individual bones within this universal set of connection rules must each be evaluated for how it would should connect. This is because of the variety in the design of the bones in the skeletal structure and the functions they serve. Because there is no real world equivalent for combining parts of different animals, there is no data to draw upon. Bone connection rules were therefore created by observing the effect of size change within animals of the same species and consultation with Dr. Jeremy Wasser. As patterns emerged in the connection rules that defined expected deformation methods, bones were grouped into connection types, as listed in Connection Types (table 2).

3.2.3.1 Limb Bones

Limb bones consist of bones found in the limb, as defined by a shaft with joint connections on either end. In order to connect limb bones to another bone, the shaft is scaled in length, such that the ends of each bone connect at the joint. In order to maintain the structural stability of the shaft after lengthening, it is scaled in diameter based on its change in length.

Once the ends of each bone have met, the end of the connecting bone is reoriented to match the alignment of the default joint connection. The end of the connecting bone is similarly scaled to that of the default connection. By mimicking the

default structure of the joint, the joint should continue to be mechanically sound (Figure 6). As this connection does not take into account the orientation of the shaft, it is possible for the limb to lose its functionality if the designer connects at an unnatural angle. As this system seeks to guide the designer rather than restrict, this is allowed, and left to the designer's eye to prevent.

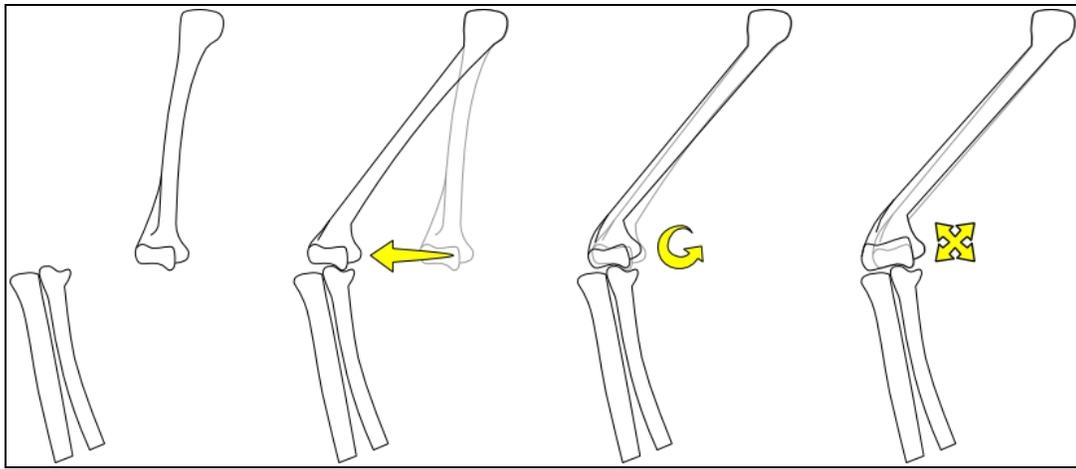


Figure 6. Standard Limb Bone Connection Steps

3.2.3.2 Scapula

When connecting to the humerus in a chest to uparm connection, the scapula may have a large gap to cross in order to create the joint connection. While the scapula does have some sense of a shaft, as with limb bones, having the shaft scale out to meet the humerus in position would likely alter the mechanics of the shoulder system too greatly. Rather than stretching the bone from its current position, the scapula will first translate rigidly to match the position of the humerus' default scapula connection.

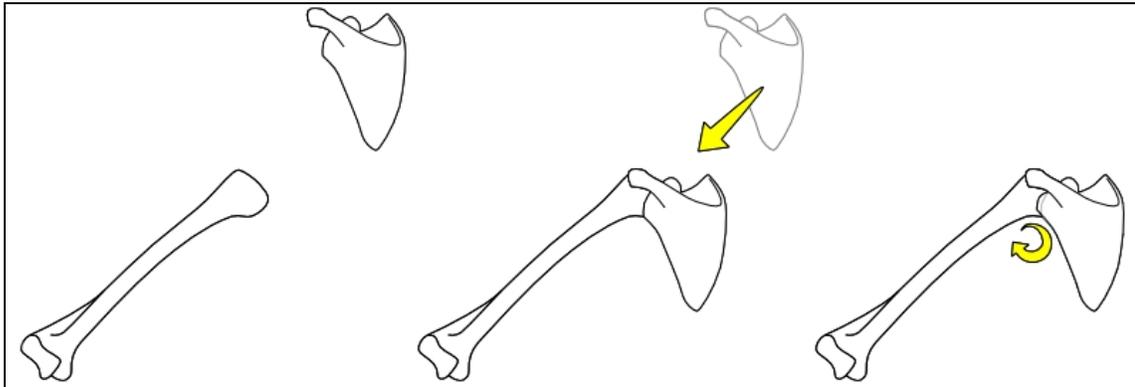


Figure 7. Standard Scapula-to-Humerus Connection

After the scapula has been positioned to minimize the distance it would have to stretch to attach to the humerus it will then accommodate any small difference in distance to the connection joint by stretching along its shaft. It finally orients to match the orientation of the humerus's default connection to maintain joint mechanics (Figure 7). In all other chest connections, the scapula simply rigidly transforms in order to maintain its spacial relationship with the ribs, should the vertebra move.

3.2.3.3 Vertebra

When connecting parts that contain vertebra, very little shape change is applied to individual vertebra. It has been observed that amongst tetrapoda, there is variation in the number of vertebra an individual animal may have. This has been extended to assume that as the vertebra in the designed creature lengthens, additional vertebra can be added, rather than lengthening the individual vertebra bones to make up for the change in length, as is done with limb bones. This allows the spine to retain its flexibility, as longer vertebra would impede this.

In order to place these additional vertebra, each vertebra in the two connecting parts is used to construct a bezier curve. The two bezier curves are then combined and additional vertebra are placed on the empty portion of the curve, while existing vertebra are repositioned to lie on the curve (Figure 8). This method does well to automatically replicate the “multi-bow model” that Hildebrand uses as an “analogy for the support of the body.” [Hildebrand et al. 1998]

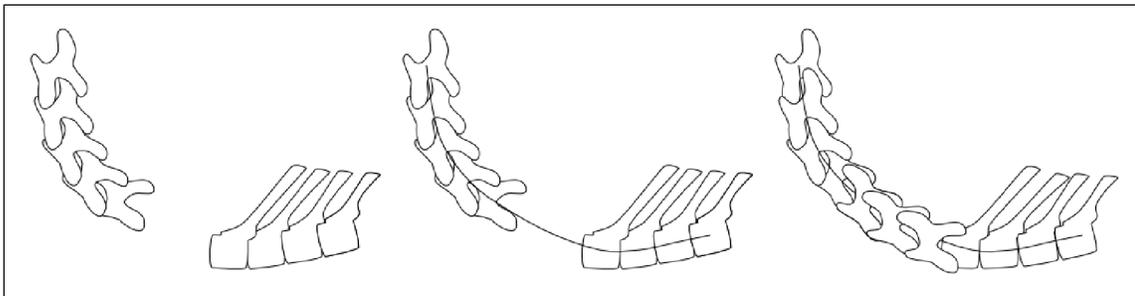


Figure 8. Standard Vertebra Connection Steps

3.2.3.4 Head Bones

Head bones consist of the skull and the mandible. Deformation of these bones to create a connection is undesirable, as a skull would likely not lengthen in any one dimension. The options for deformation would therefore be either a uniform scale or a uniform change in position. However, it is assumed that either of these deformation types would likely not match the designers intention (Figure 9).

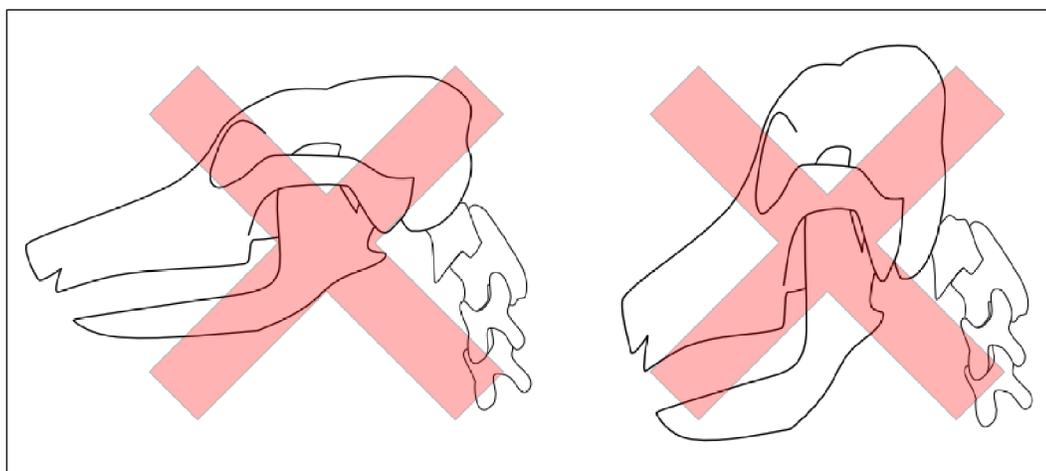


Figure 9. Undesired Uniform and Single Axis Skull Deformation to Connect to Vertebra

A change in a uniform scale to connect to the cervical vertebra would likely grow the head beyond the artists intentions for all but the smallest gaps between neck and head. Alternatively, shifting the skull and mandible in position would move the head from where the artist has deliberately placed it. Therefore, the skull and mandible never constitute an outgoing connection, but are rather always connected to (Figure 10).

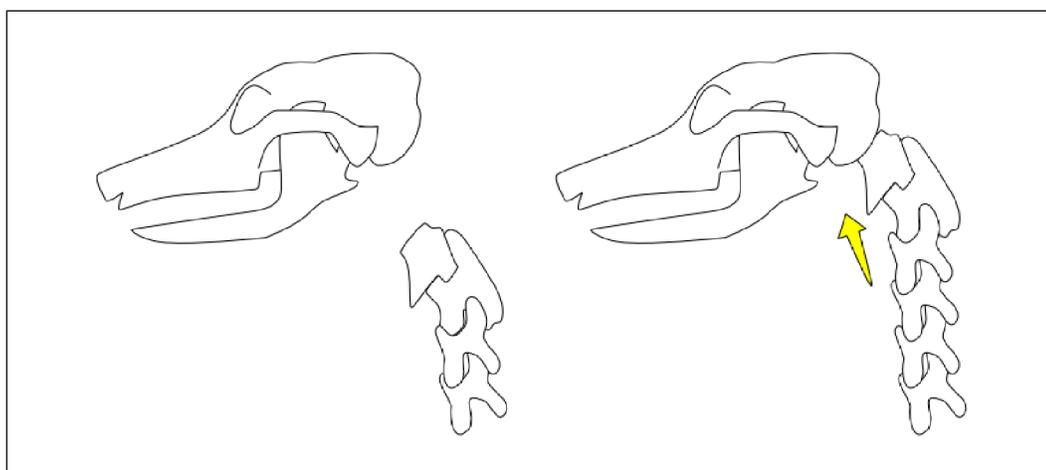


Figure 10. Standard Skull Connection Steps (Realized as a Vertebra-to-Skull Connection)

3.2.3.5 Ribs

Ribs will not actively connect to any outside part, however they must connect to the vertebra and scapula within the chest as these bones make their connections. In the case of the a vertebral connection, a rib will first rigidly move position to remain connected to the vertebra. The bulk of the rib will then maintain its orientation relative to the chest. However, the end where the rib connects to the vertebra will rotate to match any change in orientation to its associated vertebra.

In the case of a connection to the scapula, the ribs will stretch out along the transverse plane to match the scapula, without moving in the dorsal or sagittal planes (Figure 11). This allows the ribs to grow in width, without being affected by any other translations the scapula has undergone. This allows the ribcage to maintain its overall structure without warping. As there is no actual bone-to-bone connection between the scapula and ribs, it only important to maintain a close position, such that they could be connected by musculature.

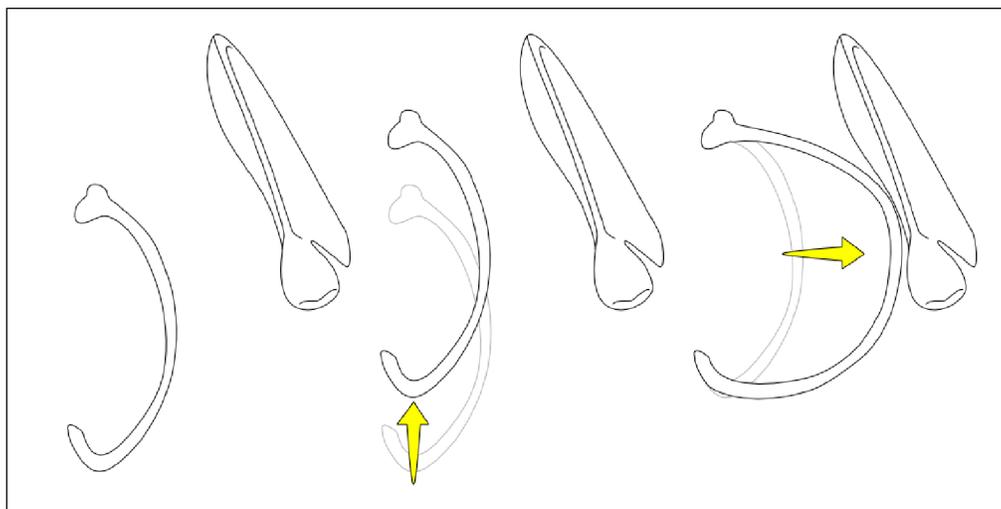


Figure 11. Standard Rib-to-Scapula Connection Steps

3.2.3.6 Pelvis

When the abdomen part connects to a thigh, the primary connection will be pelvis to femur. In this case, the pelvis will first attempt to get as close to the femur as possible while keeping its center point on the sagittal plane. This will ensure that the pelvis does not drift to one side and remains centered with the body. The pelvis will then stretch out along the remaining non-sagittal axis to meet with the femur, much like the ribs deform (Figure 12). This stretching should preserve the shape of the ilium and sacrum occurring around the area of the sacroiliac joint.

Because there is no effort to maintain symmetry within this system, to preserve artistic freedom, it is important to note that asymmetrical connections to left and right side is discouraged, as this compromises efforts to create believable creatures. This is true of all connections to sided parts, but especially true of abdomen to thigh connections. This is due to the fact that the pelvis is the only bone in the skeletal system that is a single bone that connects at both the left and right side. As such, an asymmetrical connection can yield especially strange looking results in terms of the bone's form.

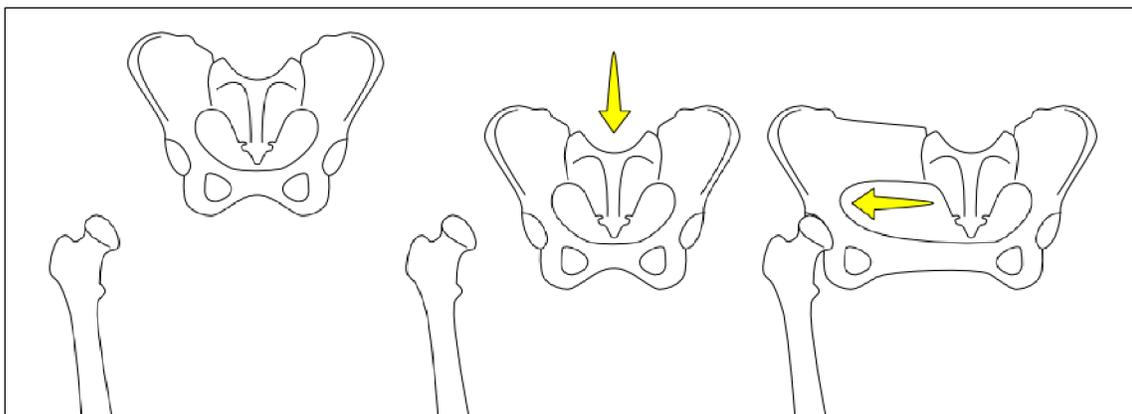


Figure 12. Standard Pelvis-to-Femur Connection Steps

3.3 Allometric Scaling

In the course of combining various animal parts together, it is likely that bones will change in length. This change in length constitutes a change in scale, which has, “structural and functional consequences.” [Schmidt-Nielson 1984] In addition to supporting the weight of the animal, bones must also endure forces applied through locomotion and impact with other objects. As this system is attempting to maintain the structural and mechanical stability evolved in nature to guide a more believable creature, it is important to attempt to compensate for the consequences that result from changing scale.

Schmidt-Nielson observes that the three parameters that can be changed to deal with these consequences when the size of a structure is altered are dimension, material, and design. [Schmidt-Nielson 1984] The overall design is dictated by the user, and it is the intent of the system to avoid interfering with this as much as possible. The lower-level designs established by the skeletal anatomy of the creature are preserved by

mandate of the system, so altering design is not an option. While one could claim that the parts in question are of a different material besides bone and tissue, and change this element, this would detract from the attempt at realism based on what the viewer is familiar with and is also unacceptable. This leaves the system to change the dimensions of the bone, changing width in proportion to length.

Allometric study is a field that attempts to examine how the various aspects of animal anatomy relate to size and scale and is therefore an ideal field to draw guidance in applying this change in width. The study of allometry is a data-driven field, measuring similar organisms and recording various aspects of their size. By comparing similar organisms of a wide enough variety of size and shape, certain allometric equations have been derived. These allometric equations are considered descriptive, rather than biological laws, but they are “useful for estimating an expected magnitude for some variable.” However, it should be noted that they “cannot be used for extrapolations beyond the data on which they are based”. [Schmidt-Nielson 1984]

While this fact limits the usefulness in predicting dimension change in animals outside the data-set for scientific goals, such equations should be more than adequate for providing scientifically-based guidance in the design process. It has been found that “interspecific comparisons of mammals and birds show that skeletal allometry is modest, with most groups scaling ($l \propto d^{0.89}$)”. [Biewener 2005] While there is scholarly disagreement on the exponential value in this proportion, this general equation will be used to guide diameter changes in bones as their length changes when implemented in this system.

4. SYSTEM IMPLEMENTATION

4.1 Technical Specifications

The beastMaster system created for this thesis was developed on a number of different computers and operating systems. Most recently, the system has been developed on a commercial-grade laptop with a Intel Core i5 Quadcore 2.5ghz Mobile processor, with an available 4gb of RAM, running a Windows 7 64 bit operating system. This is a fairly average, if not underpowered system, and so the tools performance can be considered indicative of the average user's experience.

The tool was developed as a script tool in Maya 2009. The scripting language utilized was Maya's integrated Python 2.5. Python integration into Maya 2009 was still fairly new, and as such, there is incomplete support for Maya commands. In order to deal with this, there are limited instances in the tool where elements had to be implemented in MEL rather than Python.

Python was chosen as the primary scripting language of choice due to its use in most 3D software packages such as Maya, 3D Studio Max, and XSI. While this decision would make it easier to port to other software packages, it would still be a difficult process due to the script's heavy reliance on the Maya command library and the occasional use of MEL within the tool.

Beyond the standard Python libraries and the aforementioned Maya library, the tool relies on very little outside code. The only other Python library heavily relied upon is the Euclid library, which offers a basic implementation of a vector type. This type

offers standard vector functions such as normalization, length retrieval, dot and cross product, and basic algebraic functions. Other, more advanced, vector math functions had to be created custom for the tool.

In addition to those few, select Python libraries, the tool relies on the MayaMuscle plugin, which was standard to most versions of Maya 2009. The system does not make use of the muscle system included in the plugin, but rather a single ray casting function that was not available through the standard Maya command library, but available in the Maya API. The scope of this thesis precluded development within the Maya API, so this plugin was utilized as a workaround for that limitation.

4.2 User Interface

The first half of the UI that the user will interact with is the beastMaster window (figure 13). This interface is built on Maya's default User Interface Tools, and as a result, is a less flashy user interface than more modern QT options provide. However, given the limited function the window must perform, a simplistic user interface is more than sufficient. The window predominantly contains drop-down menus called Part Selectors. The 17 part selectors each correspond to a specific customizable part defined by the beastMaster system. Each Part Selector, in addition to the default None, has a list of all installed animals that have the specific part the selector is associated with.

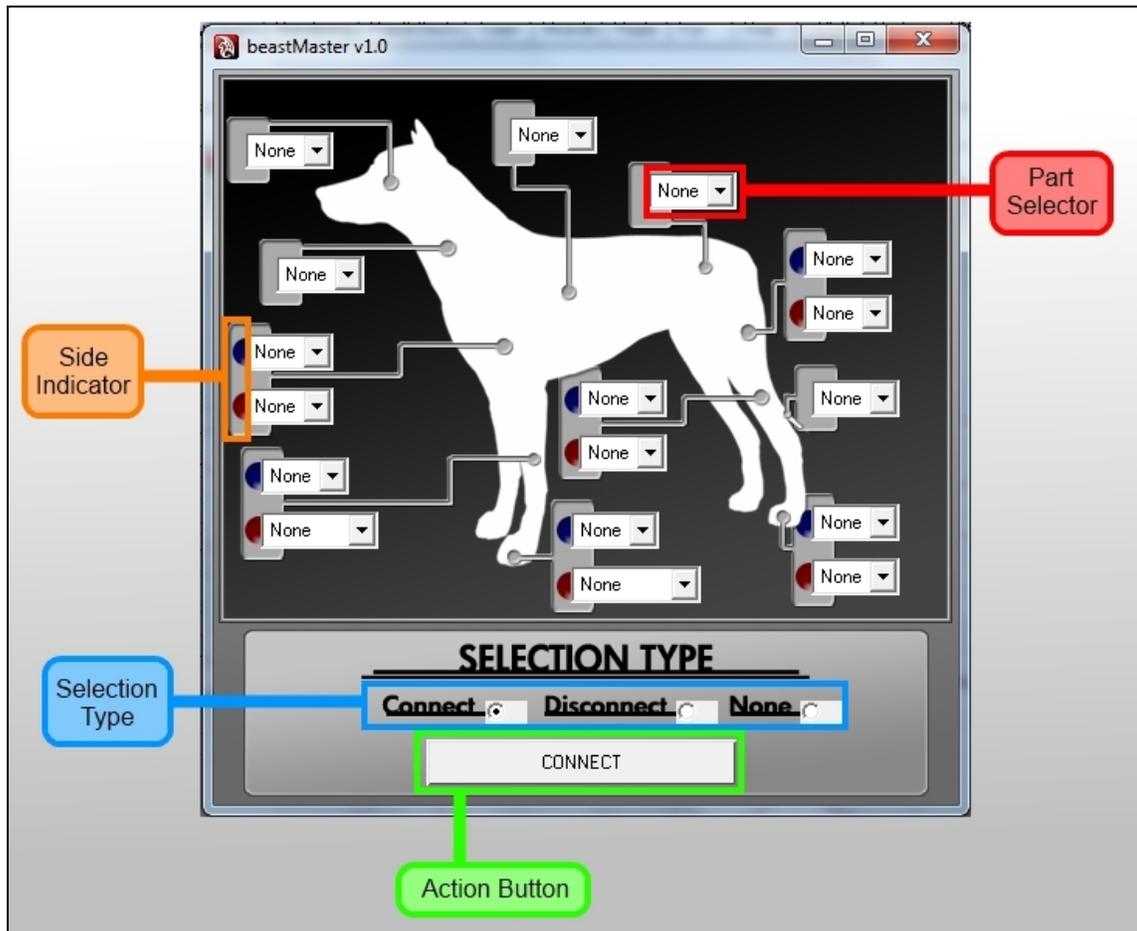


Figure 13. The beastMaster Window

Part Selector association is indicated to the user by a line connecting each Part Selector with a part on the silhouette of a dog featured at the center of the window. The choice to use a dog as silhouette was largely an arbitrary design choice and has no bearing on the actual animal selected in any given Part Selector. Because the silhouette used in the image is two dimensional, it was difficult to represent both left and right sides for some parts. Therefore, each left/right sided Part Selector pair shares a single part indication line. Their sidedness is then differentiated by colored half-circles preceding

the Part Selector. Because the part indication lines are all connected to the left-sided parts, the left-sided Part Selector is listed first on the list and is indicated with the blue circle, while the right-sided Part Selector is indicated by the red circle.

In addition to loading parts, the UI provides the method of enacting the main actions of the system. The action is determined by the Selection Type and is enacted by the modal action button. In addition to Connection and Disconnection, the None Selection Type is available to deactivate the augmented selection process in the Maya 3D Viewer, returning selections to their standard behavior.

Transformation of the parts within the Maya 3D Viewer are considered the second half of the beastMaster UI (Figure 14). In conjunction with standard Maya transformation tools, the beastMaster tool creates the aforementioned background process that augments standard Maya selections. Most notable to the user is the application of material colors to selected parts to provide visual feedback to the user indicating selection order, which is important to the connection process.

In addition, the augmented selection process is necessary because it will generally be the case that the user is selecting the skin mesh that is visible in the 3D view. Transformation of the skin would result in the other elements of the part being left behind, thereby breaking the system. In order to ensure all part elements are transformed uniformly, the part's topmost parent group must be selected and transformed. The augmented selection process ensures that when an element of the part is selected, the selection is changed to the correct group.

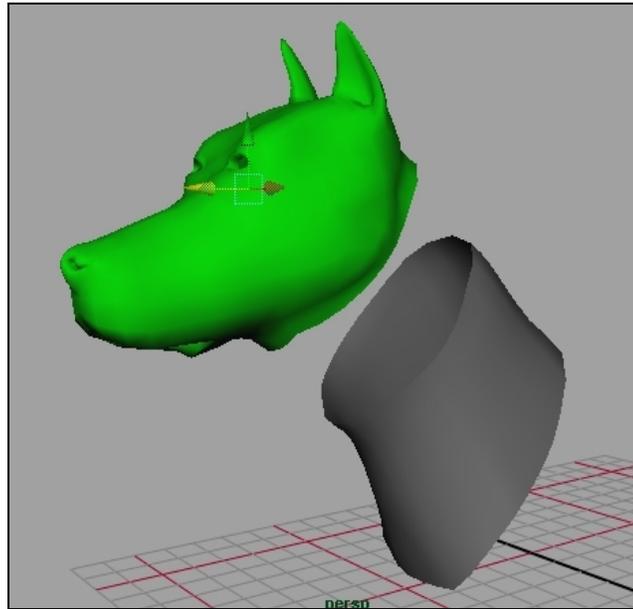


Figure 14. Transforming Parts Within the Maya 3D Window

Secondly, following the connection of parts, they continue to exist as two discrete parts, despite appearing connected in the 3D Viewer. In order to ensure they retain their virtually connected relationship, the augmented selection will select all parts connected to the element selected, in addition to its parent group. In this way, these parts are all transformed as though they were a single entity.

4.3 Part Connection

When the connection process is initiated via the Connect Action Button, the system begins by determining the selected part groups based on their assigned shaders, as set by the augmented selection system, leveraging the selection filtering that has already been done during shader assignment. From the two selected part groups, the script determines which part in each group will be connected. Because the system

defines previous connections virtually to maintain part structure, rather than actually merging them in to a single part, only two parts in the opposing part groups will actually connect. If no valid connections are available between the two part groups, the user is warned and the connection process is halted. If a valid connection is discovered, the script first updates its virtual connections list to indicate that these two parts are now a single connected part.

4.4 Bone Connection

Once the virtual connection has been established, the script begins to deform the bones to a visually connected state. The Connection List File corresponding to the out part (the connecting part) is read to find the bone-to-bone connection list relating to the current in part (the connected-to part). If the part is of Manus or Pes type, the script will now branch to a special bone deformation method. Otherwise, each bone-to-bone connection in the acquired connection list is iterated through, connecting each bone one at a time.

4.4.1 Default Bone Deformation

At the beginning of the bone deformation process, the joint structure used to deform the bone are dynamically generated. During this process, the type of connection required is determined. There are nine different connection types (Table A-2), which determine the process by which bones are connected. An effort was made to keep these types as generalized as possible to minimize branching in the bone deformation code. However, many types of connections require special procedures for proper deformation. As a result, while some types are generalize to a large number of different possible bone

to bone connections (ex. type 1), some connection types describe a unique connection between two specific bones (ex. Type 8). Based on the currently selected type, the script branches to one of the four distinct bone deformation processes described below.

4.4.1.1 Type A Deformations

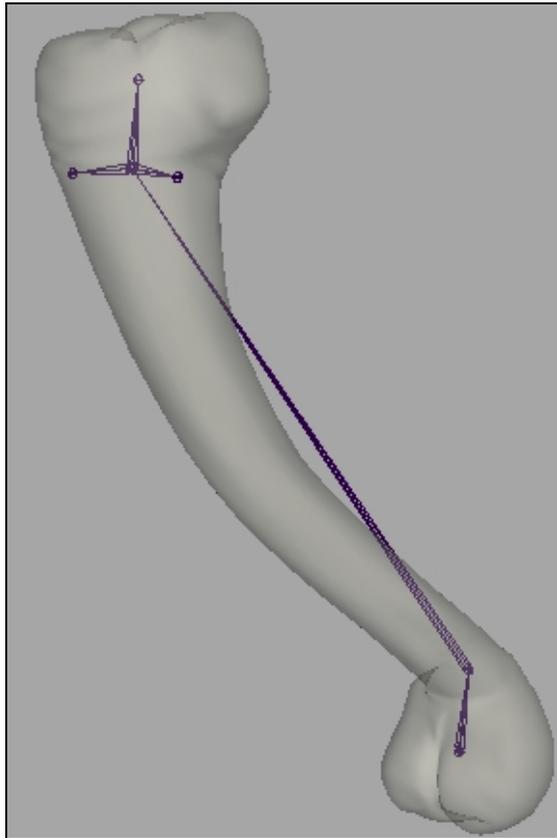


Figure 15. Typical Type A Deformation Joint Layout

Type A deformations (Figure 15) are used to connect bones of the limb (types 1 and 2). This connection involves the start of the bone remaining in place, while the shaft of the bone stretches so the end reaches the corresponding connection point locator

contained in the in bone. The connection bone joint of the out bone then orients to alignment point point locators, also contained in the in bone.

Type A deformations are processed for connection types of both type 1 and 2. In the case of a type 2 connection (scapula to humerus), an extra process is applied before the rest of the deformation takes place. This process briefly restructures the joint chain, so the shaft end joint is the root of the joint structure. The shaft end joint is then immediately snapped to its connect point, ensuring the entire scapula rigidly translates into position. After this snap, the original joint parenting structure is re-established. This extra step ensures the shaft area of the scapula does not stretch as the process continues.

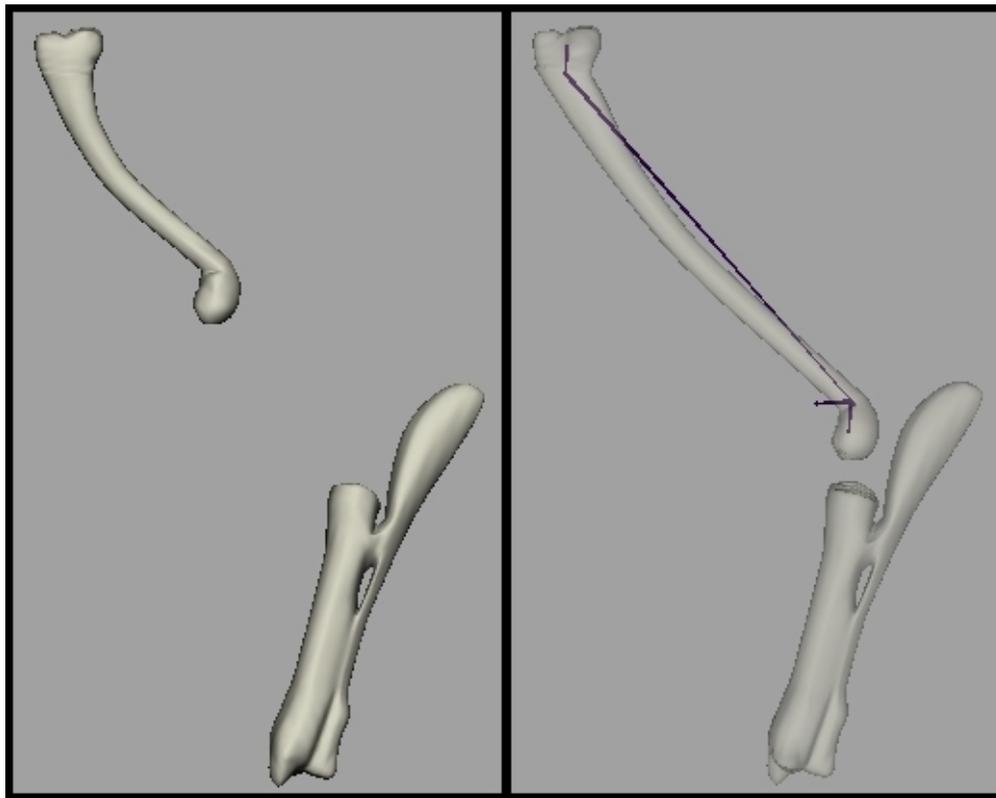


Figure 16. Limb Bone Connection by Length Scaling

The length ratio is determined by dividing the distance from shaft base to shaft end by the distance from shaft base to the connection point locator. After aiming the axis of the shaft base joint that points down the length of the shaft at the connection point locator, this ratio is applied as a scale value along this same axis. Scaling is applied to the shaft base joint (Figure 16) to lengthen the shaft, rather than translating the bone end to the connection point to support the rigid binding applied during setup. This allows the shape of the shaft and vertex distribution to be maintained during lengthening, without necessitating a more complex vertex binding between base and end joints.

Once the shaft has been scaled, such that the shaft end joint is coincident with the connection point locator, the end of the bone must be scaled and oriented to maintain the mechanical stability of the anatomical joint connection between bones (Figure 17). In order to scale the shaft end joint to the required size, the distance from connection point locator to one of the alignment locators is divided by the distance from shaft end joint to one of its child alignment joints. This results in a scale value uniformly applied to all axis of the shaft end joint.

Orientation is enacted as a two-step process that requires the shaft end joint be aimed at the first alignment joint. As this is not the case and the shaft end joint cannot be reoriented without disrupting the bone vertices bound to it, a new joint is placed coincident to the shaft end joint, aimed at the first alignment joint, and then inserted into the structure as parents of the alignment joints. This new joint is then aimed at the first alignment point locator, rotating the first alignment joint into a matching position.

In order to rotate the second alignment joint into place without disturbing the

position of the first, the signed angle between the vector of shaft end joint position to second alignment joint position and shaft end joint position to second alignment locator position is found. This angle represents the amount to rotate the shaft end joint. The axis to rotate around by this amount should be the normal of the plane defined by these three points. The aimed axis of the newly created joint is equivalent to this normal, allowing the angle to be applied as rotation around this axis.

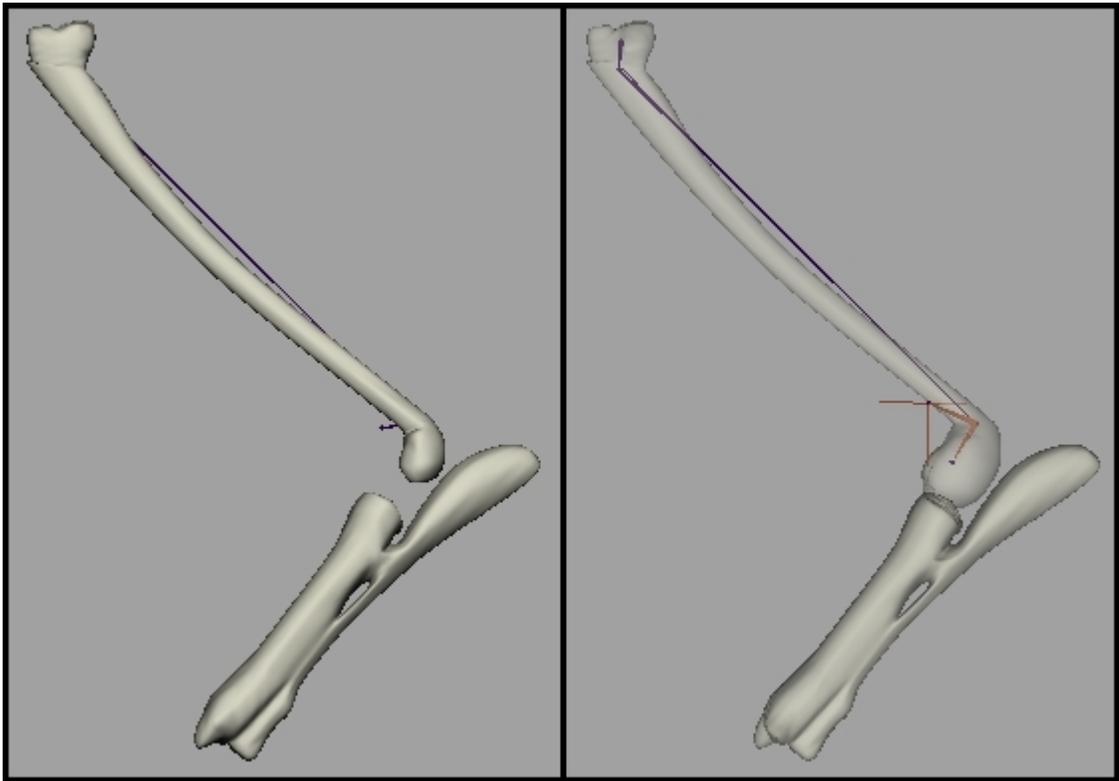


Figure 17. Matching Orientation and Scale of Connected-to Bone

4.4.1.2 Type B Deformations

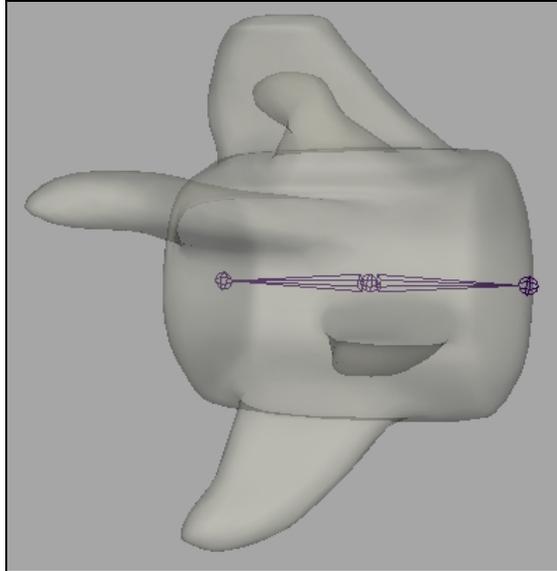


Figure 18. Typical Type B Deformation Joint Layout

Type B deformations (Figure 18) are applied to connections involving vertebra. Unlike other deformation types, this process adds extra bones, in addition to deforming existing vertebra. Also unlike other deformation types, a type B deformation deforms all connecting vertebra in the function, rather operating on a single bone like most deformation types.

Like other deformation types, the bone connection is dictated by an entry in the bone connection list. In the case of Type B deformations, the entry is stored as a single generalized description of the vertebra-to-vertebra connection (ex. ChestVertebra_F-NeckVertebra_B), rather than a complete numbered and ordered list of all the vertebral connections that will occur. A more complete connection list is built on the fly before

deformation. With the full list of connections, each vertebra's Bone Attribute file can be read, allowing the generation of each vertebra's deformation joints.

When deformation joints have been created for all connecting vertebra, curves used to describe the shape of the vertebral columns for the out and in parts are created. For each part, a curve is created that travels through all its vertebra. This curve is generated through Maya's curve creation operation, supplied with an ordered list of the positions each vertebra's root deformation joint. If the part is the head, then a one degree, two point curve is created, as there are only two available positions describing the base of the skull and the tip of the nose. These two part curves are then joined with Maya's curve attachment operation to create a third vertebra curve (Figure 19) to define the new flow of the vertebra upon connecting out and in parts.

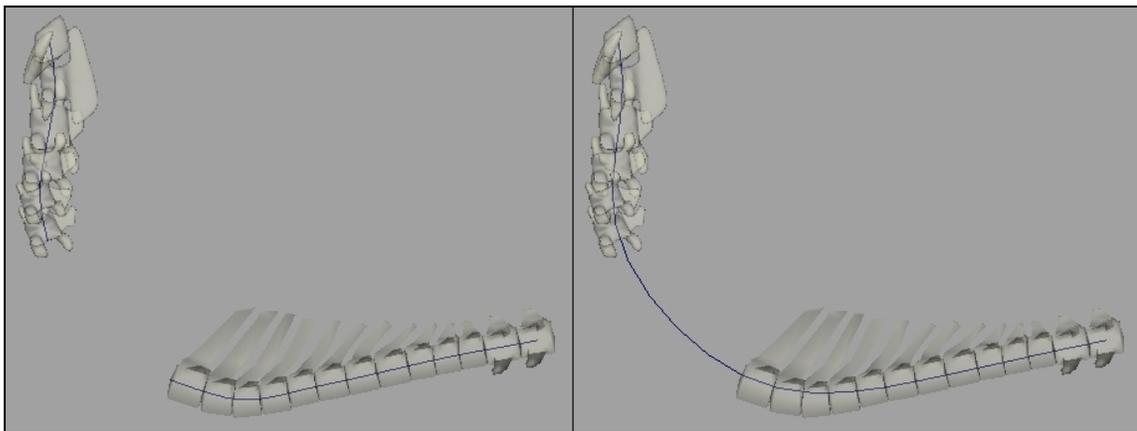


Figure 19. Part Curves Generated Through Vertebra and Joined to Create a Vertebra Curve

If the distance between out and in parts has grown, the length of this new attached curve will be longer than the existing vertebra can cover. In this case, extra

vertebra are created to fill gap. The number of extra vertebra required is found by dividing extra length of the vertebra curve by the length of a single extra vertebra.

The length of a single extra vertebra is found by first determining which vertebra will serve as a template for creating extra vertebra, using information created at setup time and retrieved from the Bone Attribute File. This template vertebra's length is dictated by the distance between its deform joints 1 and 3, which are located at either ends of the vertebra.

The length of the segment of the vertebra curve where extra vertebra should be placed is defined as the distance down the length of the curve where the out vertebra curve ends to where the in vertebra curve starts. These points are determined by the closest point from the last out/first in vertebra to the vertebra curve. As Maya has no built in closest point on curve operation, a `closestPointOnCurve` function was created to step along the curve by a specified number of divisions and find the distance from the point on the curve at each step to the input point. While a linear, start-to-end search is not a very efficient approach to calculating the closest point, it works well enough given the limitations of the Maya scripting language. This function returns the arclength at these two points, the difference of which defines the extra segment length.

The prescribed number of extra vertebra are then created, each as a fully functional bone, complete with its own locators and Bone Attribute file. This allows these extra vertebra to behave as any installed bone would, ensuring that they continue to function with the system in future deformation. With the requisite number of vertebra available to fill the vertebra curve, all out, extra, and in vertebra are placed along the

vertebra curve in order.

Vertebra are positioned along the curve by snapping their root deformation joint to a position evaluated from the vertebra curve based on a prescribed length. The prescribed length for each vertebra found by adding half the length of the current vertebra to the sum of the lengths of all previously added vertebra. In order to then retrieve the position from this length value, it is necessary to use the custom `findDistDownCurve` function, as Maya's scripting language only supports the evaluation of position on curves based on a parametric value u . This function uses a bisection search method on the curve in order to find the u value that corresponds to the input distance down the curve. With the set precision of 0.01, it generally takes around ten steps to find the desired position, so despite the iterative nature of this search, it does not take long to return a result. The resulting u value is then used to evaluate the worldspace position, which the root joint is moved to.

After the root joint has been positioned on the curve, it is oriented by calling the `findDistDownCurve` function a second time, with half the length of the vertebra added to the previous length argument. The root joint is then aimed at this position, orienting it to be approximately tangent with the vertebra curve (Figure 20).

Because the extra length may not be perfectly divisible by the length of extra vertebra, spacing may need to be adjusted to allow all vertebra to cover the vertebra curve from start to finish. This extra spacing is applied as a multiplier on the measured length of each vertebra. To find this multiplier, the total length of all vertebra are summed and compared to the arclength of the vertebra curve. Because this value will

not typically result in a very big change in spacing when distributed amongst all vertebra, no effort is made to stretch or compact the vertebra to compensate, conforming to the decision that individual vertebra size remain consistent through deformation.

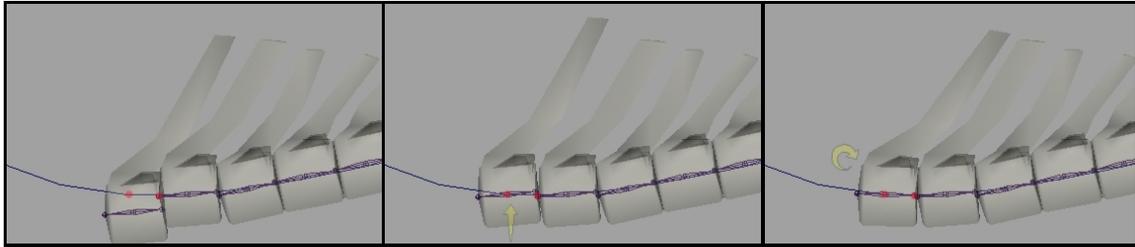


Figure 20. Vertebra Placement and Orientation Along Vertebra Curve

4.4.1.3 Type C Deformations

Type C deformations (Figure 21) are specifically for connecting ribs found in the torso to the scapula by stretching the shaft of the rib horizontally to connect to the scapula. Because this horizontal stretch does not occur linearly through the length of the rib, the standard method of stretching bones by scaling out rigidly bound vertices is not effective. Rather, this type of deformation is unique in its use of a smooth binding in order to perform its bone deformation.

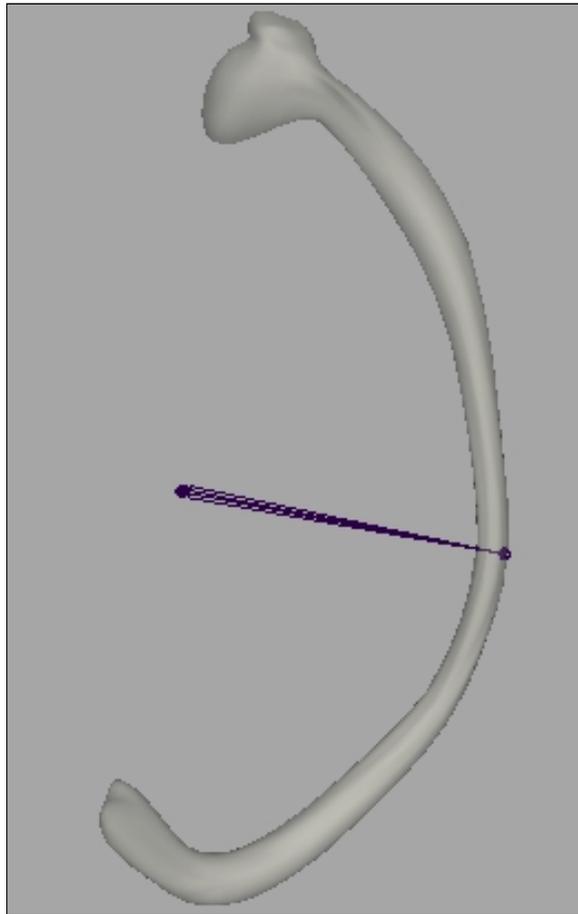


Figure 21. Typical Type C Deformation Joint Structure

Before deformation, the distance required to stretch the ribs horizontally to meet the position of the scapula is found. In a general sense, this is the difference in the x-axis between the rib's root deform joint and the corresponding connection point contained in the scapula group. However, it cannot be assumed that the torso will always be aligned to worldspace, so this distance needs to be calculated in the current local space of the torso.

The current local space is calculated based on additional locators created at setup time and stored in the rib group, named `pivPos`, `ribDir`, and `headDir`. The vector from

pivPos to ribDir generally describes the direction the rib shaft is facing in the part's local space. The vector from pivPos to headDir similarly describes the direction to the head. These three locators describe a plane that is parallel to the torso's sagittal plane. While local space could potentially have been derived from the local space xform stored in the part group, these locators ensure the user cannot break the local space by freezing transforms.

In order to calculate the vector along which the rib stretch will occur, the worldspace position of the first rib's root deform joint is found, followed by the worldspace position of the scapula connection point. While most bones in the system have a corresponding connection point suffixed with the specific bone name (ex. RtRib01), in this case there is only a single connection point for all ribs, which uses a generalized bone name (ex. RtRib).

Next, the normal of the out part's sagittal plane must be found. While the sagittal plane itself is not accessible, the first rib contains the aforementioned locators that define a parallel plane. The cross product of the vectors from pivPos to ribDir and pivPos to headDir produces the normal of the plane. With this normal and the two positional vectors already acquired, the closest point on the plane defined by the normal and the connection point position is found.

Because the objective is to only allow translation in the local-space x-axis, the target point is calculated as the projection of the first rib's designated deform joint onto a plane parallel to the sagittal plane, running through the connection point (Figure 22). The vector from deform joint to target point defines the translation that will be used to

stretch all ribs found in the out part.

Next, a list of all groups that match the generalized bone name is acquired to iterate over and apply the newly obtained stretch vector to. For each successive rib, the deformation joint chain for the bone is built and the same stretch vector is applied to the end deformation joint in the chain. The rib shaft is bound to this end joint, with a smooth binding falloff towards each end of the rib. This smooth binding allows a stretch to occur evenly through translation rather than scale.

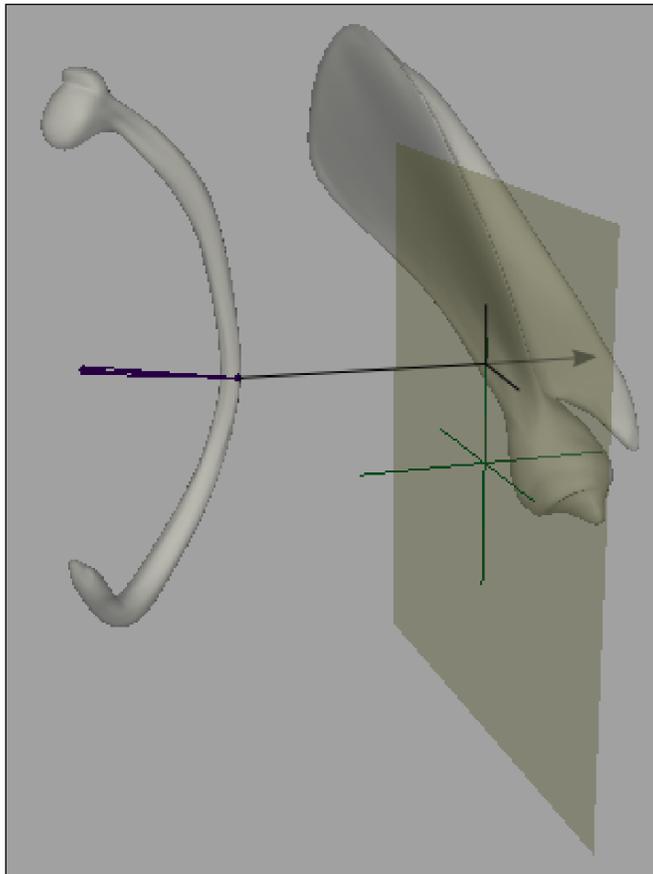


Figure 22. Determining Target Point for X-Axis Stretching

4.4.1.4 Type D Deformation

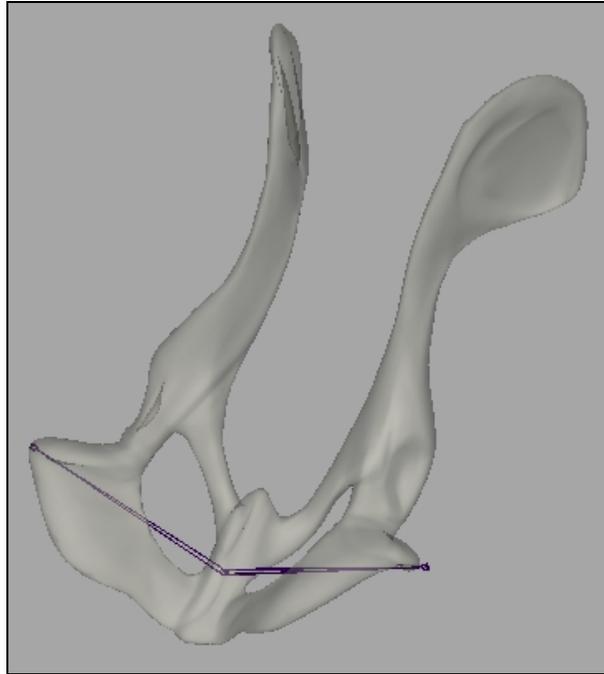


Figure 23. Typical Type D Deformation Joint Structure

Type D deformations (Figure 23) is used to shift bones in any axis except horizontally. In this way it is very similar to the Type C deformation, although ignoring the opposite set of axis. Unlike Type C, these deformations do not use any deformation joints to achieve their transformation. Rather, transformations are applied rigidly to the entire bone mesh.

The transform vector used in a type D deformation will always be the same for all bones that require a type D deformation within a single part connection, as type D deformations following the first are merely simulating an attachment to that first bone. Therefore, as an optimization, the movement vector calculated at the first bone in the part is passed to successive type D deformations. If this vector has been previously

calculated, the type D deformation will use this vector, rather than recalculating it.

If no transform vector has been previously established, a vector to align the root deform joint with the local space y and z axis of the connection point is calculated. In order to get this vector, a target point is calculated as the projection of the connection point onto a plane parallel to the sagittal plane, running through the designated deform joint. The vector from deform joint to target point defines the transform vector that will be used to translate the out bone.

Once the movement vector has been established, the deformation joints created for the in bone are no longer necessary, as they were only used to positional information. Furthermore, this joint structure will get in the way of translation applied directly to the bone mesh. Therefore, all deform joints are deleted before the transform vector is applied. After the transform vector is applied to the bone mesh, it is returned for use in any remaining Type D deformations.

4.4.2 Special Case Method

The bones in Manus/Pes type parts are all type 1 connections, utilizing type A deformations. However, before processing the type A deformations, all bones must be repositioned relative to all other bones in the part. Although this can be considered a pre-processing step prior to enacting a type A deformation, the number of actions required in the repositioning are so robust that this was broken off into a special case (Figure 24).

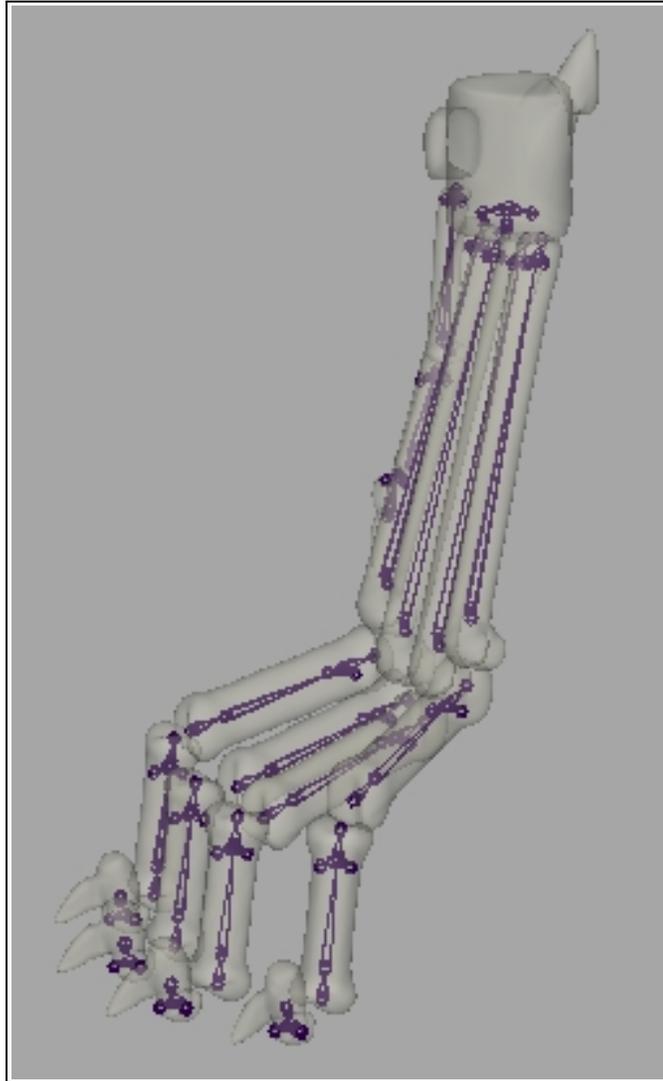


Figure 24. Typical Special Case Deformation Joint Structure

The repositioning prior to deformation is determined by finding parametric values describing each bone's relative position between start and pre-end points and then applying these values to find the equivalent position between start point and the post-end points. The start point, which remains consistent through repositioning, is defined as the average of the root deform joint positions of all non-thumb phalanges. The pre-end

point is found to be the position of the end deform joint of the bone that will be connecting to the opposing part, while the post-end point is found to be the position of the connection point locator in the opposing part.

When the necessary positional data is found, the parametric values can be found and applied. This is achieved by taking the position of the input joint and determining the percent it lies between start and pre-end point for each xyz axis, yielding a triple set of parametric values. The formula to find these xyz values is then reversed, with the new post-end point substituted in, in order to determine what the new position of the object should be (Figure 25). The object, in this case the base deformation joint of each bone, is then moved to this position. In this way, each bone is rigidly translated into its new relative position.

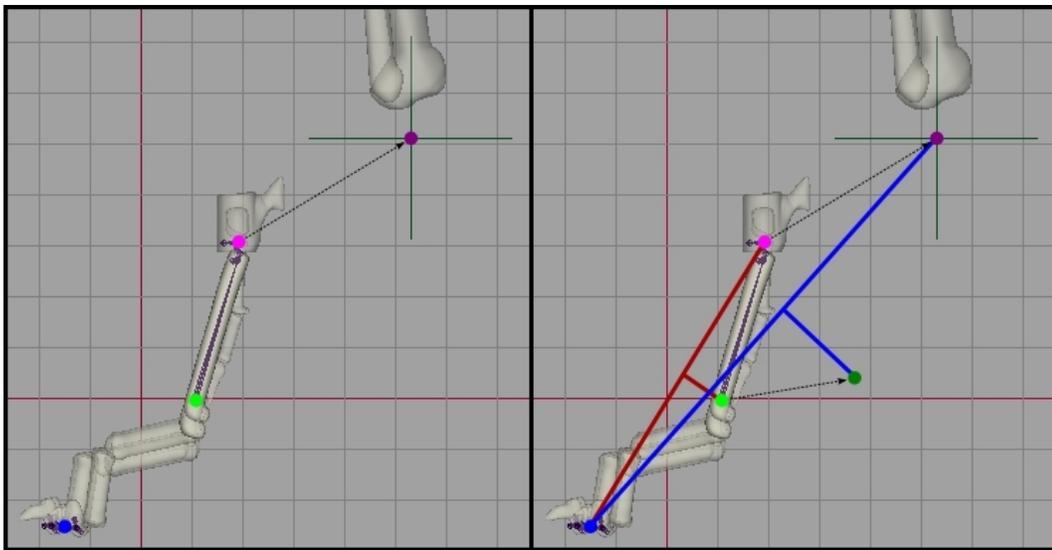


Figure 25. Relative Positioning. (key: Bright Green=Input Point, Blue=Start Point, Pink=Pre-End Point, Purple=Post-End Point, Dark Green=Output Point)

Although each bone has been correctly positioned, the rigid translation means they are not yet connected to their in bones or properly oriented. In order to achieve this, each bone is connected to the next through the standard type A deformation. The type A deformation lengthens or shortens each bone as necessary and ensures they are properly oriented to the new spacing.

4.4.3 Allometric Scaling

When lengthening bones during the deformation process, it is important to increase the diameter as well, as described earlier. The allometric equation used by this system, ($l \propto d^{0.89}$) requires knowledge of the diameter of the bone to apply the scaling. Because the diameter of the bone is not consistent throughout its entirety, diameter is determined on a per-vertex basis. Allometric scaling based on diameter is also applied on a per-vertex basis. Although applying a uniform scale to the bone would be simpler, it is difficult to determine a single scale value due to non-uniform distribution of vertices throughout the bone mesh. Therefore, the function iterates through all vertices in the mesh, determining the bone diameter at that vertex and calculating the specific allometric scaling for that vertex.

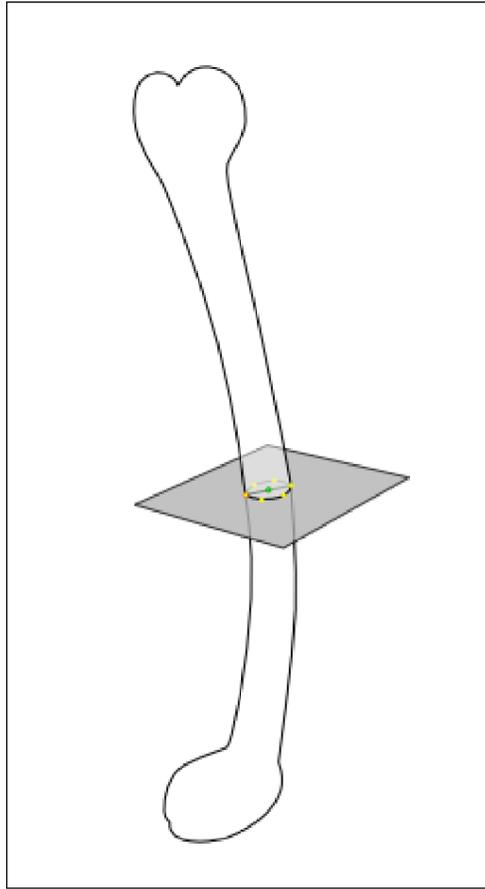


Figure 26. Procedural Center Point Determination for a Single Vertex

In order to determine the diameter of bone at a given vertex, the center point is first found by first creating a planar slice of the bone at the queried vertex (Figure 26). To create this slice, a duplicate of the bone is created, to allow slicing without changing the topology of the original bone. The duplicate bone is sliced along the plane running through the queried vertex whose normal corresponds to the vector from shaft start to shaft end deform joints, describing the shaft of the bone, which has been lengthened.

The cut action is performed through Maya's polyCut command, which creates a

new vertex at any point where an edge intersects the given cut plane. In order to determine the new points that have been created by the polyCut command, the original and duplicate bone vertex counts are evaluated. Because all new vertices are given indices at the end of the mesh's vertex list, the list of newly created vertices can be assumed to be the range from the size of the original mesh's vertex list to the size of the new mesh's vertex list.

Because the polyCut action created new points, even where points already existed, the point that coincides with the point currently being evaluated for diameter must be found. This is accomplished by running through the worldspace positions of all of the newly cut vertices and finding the distance from the queried vertex to each newly cut vertex. Because of inaccuracies in the cut, there will rarely be a perfectly matching vertex, so the closest vertex after checking the distance to all newly cut vertices will be considered the coincident point.

This coincident point is then used as the starting point to determine the vertex loop that has been cut at the queried vertex. This process ensures that only vertices defining the circumference of the bone at the queried vertex are considered. This protects against a curvature in the bone or jutting-out feature that may have intersected the cut plane injecting additional vertices into the calculations and skewing the center-point results.

Because it is unknown whether or not the points on the vertex loop will be evenly distributed, a simple average of the loop points is not sufficient for determining the center-point. Instead, the center point is calculated by finding the centroid (geometric

center) of the polygon defined by the loop points. The equation used to determine the centroid of the polygon requires a two dimensional polygonal state, so the list of vertex positions yielded from the cut is first converted into two dimensions. Because all the vertex positions are planar, having been cut along the same plane, this conversion to two dimensions is possible without any projections that could impact the results. In order to convert to a two dimensional coordinate system, two vectors on the plane of the polygon that are orthogonal to each other are found. The first vector, which will serve as the x-axis in the new coordinate system is found by finding the first pair of indices in the input vertex list that does not have a length of zero. The second vector, which will serve as the y-axis, is found by taking the cross product of the first vector and the normal to the plane.

Finally, a point on the plane of the polygon that isn't colocated with the other points is established to act as the origin. This point is calculated by finding the point halfway between the first and second points of the polygon. This is assumed a safe assumption as a non-colocated point, as the bone circumference almost certainly will not present self-intersect. Each three dimensional vertex position defining the polygon is then converted to two dimensions by first finding the offset of the vertex position from the 2d origin, multiplying this offset by the 2d x-axis, and finding the sum of the values stored in the resultant vector to get the 2d x coordinate. The same process is then done with the 2d y-axis substituted in is done to find the y coordinate, yielding the final result.

Once converted to two dimensions, the ordered list of 2d positions is passed to the `compute2DPolygonCentroid` function. This function returns the 2d centroid of the

polygon (C_x, C_y) by calculating the formula:

$$C_x = \frac{1}{6A} \sum_{i=0}^{n-1} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

$$C_y = \frac{1}{6A} \sum_{i=0}^{n-1} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

where x_i and y_i are two coordinates of the polygon and A is the polygon's signed area:

$$A = \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i) \quad [\text{Bourke 1988}]$$

The returned centroid is also in two dimensional space, and must be converted back to three dimensional worldspace before continuing. This conversion is performed by, for each xyz coordinate, taking the sum of the current coordinate of the 2d-system origin the product of the current coordinate of the 2d-system x-axis by the x coordinate of the 2d centroid and the product of the current coordinate of the 2d-system y-axis by the y coordinate of the 2d centroid. The now worldspace centroid is returned. This is in turn returned by the `getDiameterCenter` function following the cleanup of the duplicate bone. The vector along which the point will translate to achieve the scale is calculated as the normalized vector running from centroid to the current queried vertex.

Before normalizing this vector, the magnitude is found and multiplied by two to determine the original diameter of the bone at this point, referred to as d_{prev} . In order to determine the allometrically scaled diameter, referred to as d , let l be the current length of the bone and l_{prev} be the previous length of the bone. The relationship between these four parameters can be written as:

$$\frac{l_{\text{prev}}}{d_{\text{prev}}} = \frac{l}{d^{0.89}}$$

where 0.89 is the determined allometric parameter. [Biewener 2005] From this equation, we can compute the allometrically scaled diameter d as:

$$d = \sqrt[0.89]{\frac{l d_{\text{prev}}}{l_{\text{prev}}}}$$

This is then divided by two, because the scale is occurring from the center point. This new radius is multiplied by the normalized translation vector and added to the center point to get the new position. This new position is then stored in a list to be applied to the polygon once all positions have been determined. In this way, processed vertices do not alter future skew future results. Once all scaled vertex positions have been found, the positions for each are set corresponding to the list.

4.5 Skin Connection and Deformation

4.5.1 Default Method

Each vertex on the skin is deformed by following its closest point on the bone through its deformation. Therefore, once the bones have been connected and deformed, the skin has the necessary information to perform its deformation and connection. This connection begins by obtaining a list of border edges that should be unaffected by the deformation, as read from the skin attribute file associated with the out part.

For most bone deformation types, the in bones remain static. There is therefore no reason to deform the corresponding in skin. However, in the case of vertebra connections, the in bones can shift slightly as they are repositioned along the spine curve. In order to accommodate this shift, the in skin is deformed prior to the standard skin deformation. This deformation process, which will be run a second time, immediately after, on the out skin, begins by finding the number of vertices on the skin to be deformed. It then converts the bones, used to drive the deformation of the skin, into reference (bones prior to connection) and current (bones after connection) affectors of type `bmModel`.

The `bmModel` class was an experiment into speeding up the deformation process by storing frequently accessed model data such as worldspace position and point/edge relationships. It was never determined what gains were made in speed (if any) by this method as opposed to the usual method of querying Maya when information is required. If nothing else, the class makes the data easier to access, so it is not without benefit. This class could have been more widely implemented throughout the system rather than

just at the skin deformation level given more time.

In order to deform the skin by the bone affectors, a relationship between the skin and the reference bone affector is be established. This relationship is defined simply as the closest point on the bone for each vertex on the skin. A Maya `closestPointOnMesh` node is created in order to retrieve the closest position on the reference bone affector for each skin vertex, as well as the index of the face on the reference bone affector upon which the closest position lies.

As each closest position is found, it is converted to a barycentric coordinate system. This implementation of a barycentric coordinate system only supports triangular faces, so bone affector have been triangulated previously. The barycentric coordinate system assigns mass to each of the three vertices defining a triangular face, such that the given point on the face of the triangle is its center-of-mass. These coordinates allow the determination of where this point should be located after the triangle is deformed. For each point on the triangular face, the mass is calculated and stored in a dictionary whose key is the index of the vertex the mass is applied to. When finished, the three element dictionary is returned as the coordinates for this particular point.

With the barycentric coordinates in relation to the reference affector determined. The equivalent position on the current bone affector can be found by applying the mass to each point to the same face on the current bone affector. By getting the vector sum of the position of each of the three indices that make up the current face, as multiplied by their corresponding mass, the deformed position of the closest point is found. The vector from the original position of the closest point to the new position of the closest point is

the vector to be applied to the corresponding skin vertex to achieve skin deformation.

This vector is stored in a list of deformation vectors to be applied at a later time.

Before applying these deformation vectors, borders must be dealt with, which do not necessarily move solely based on the underlying bone deformations. In the case of deforming the in skin, the border should maintain the overall shape of its edgeloop, avoiding being warped as the various vertices attach to different underlying bone faces. This is not a problem in the case of the out skin, as these border points will be snapped to the in border at a later time. In order to ensure the in border retains its shape, border vertices are moved by a uniform amount, equal to the deformation vector of largest magnitude.

In addition, there are often other borders that should not be adjusted by underlying bone deformations, such as those already connected to a part. The list of static border edges are similarly converted into vertex lists. The deformation vectors that correspond to the indices in these static border lists are then set from their current value to a vector of length zero, ensuring they do not move.

Once these border cases have been addressed if necessary, the deformation vectors are iterated through and applied as relative translation vectors to their corresponding skin vectors in order to achieve the skin deformation. It bares mentioning that in enacting this skin deformation, only positional change in the bone is taken into account. This allows changes in orientation at joint connection to slide under the skin, rather than unduly affecting it, enhanced by the offset of skin to bone. As a result, some sheering can potentially occur, but has not proven itself to outweigh the benefit of

disregarding orientation.

4.5.2 Additional Deformation for Added Vertebra

Following the standard deformation by bone affectors, there is a check to see if any vertebra were added at bone deformation time. If this is the case, additional skin deformation steps need to be taken. This is necessary because deformation based on the difference in the reference to current bone affectors will not take into account additional vertebra added, as there is no corresponding reference state for the new vertebra.

Since vertebra are added along a vertebra curve, the deformation is looked at as an extrusion of the out skin, adding addition segments to the skin over the course of the extrusion. In order to extrude the border, the function will require a curve running from the center points of the part borders that describes the shape of the vertebral column for the extruded segment. This curve is created by first taking the vertebra curve used to place the vertebra bones and trimming it by the arclength of the out vertebra curve, creating a curve that starts approximately at the plane of the out connecting border.

At this point, the trimmed curve runs through the vertebra, which is likely situated towards the top of border edges. In order for the extrusion to work properly, this curve must run through the center points of the borders. The trimmed curve is therefore translated so the first cv is coincident with the center of the out border. Because the borders aren't of uniform shape, the translated curve may still not run through the center point of the in border. Any such offset will is addressed at extrusion time, rather than manipulating the curve at this point.

If the length of the extrusion is sufficiently long, there will likely need to be

multiple segments of the extrusion created to describe the curvature. It is also desired that the new segments mimic the existing resolution of the skin. Therefore, the average length from each out border point to its neighbor in the opposite direction of the connection is found and considered to be the extruded segment length. The total extrusion length is determined by the arclength from the start of the trimmed curve to the closest point on the curve to the center of the in border. This extrusion length is divided by the average segment length and rounded to get the number of divisions the extrusion will have. The average length of each segment is then recalculated from its previous value to the result of the extrusion length divided by the number of divisions, in order to ensure the extrusion covers the full length of the curve.

This extrusion could be very simply accomplished using Maya's extrusion tool, which allows for extrusion along a given mc curve. However, this tool has the unwanted effect of creating new edges to define the extruded border, rather than moving the extruded border and placing new edges in between. Since border edges are defined by specific indices in the skin attribute files, these must remain consistent in order for the system to continue functioning properly after the extrusion. Therefore, the extrusion must be implemented differently to achieve the desired results while maintaining the border indices.

In order to accomplish this extrusion, extrusion locators representing the center point of each edge loop that defines a division are placed along the trimmed vertebra curve (Figure 27). These locators are created by iterating over the number of divisions and placing them at a distance down the length of the curve that is increased by the

previously determined segment length for each step.

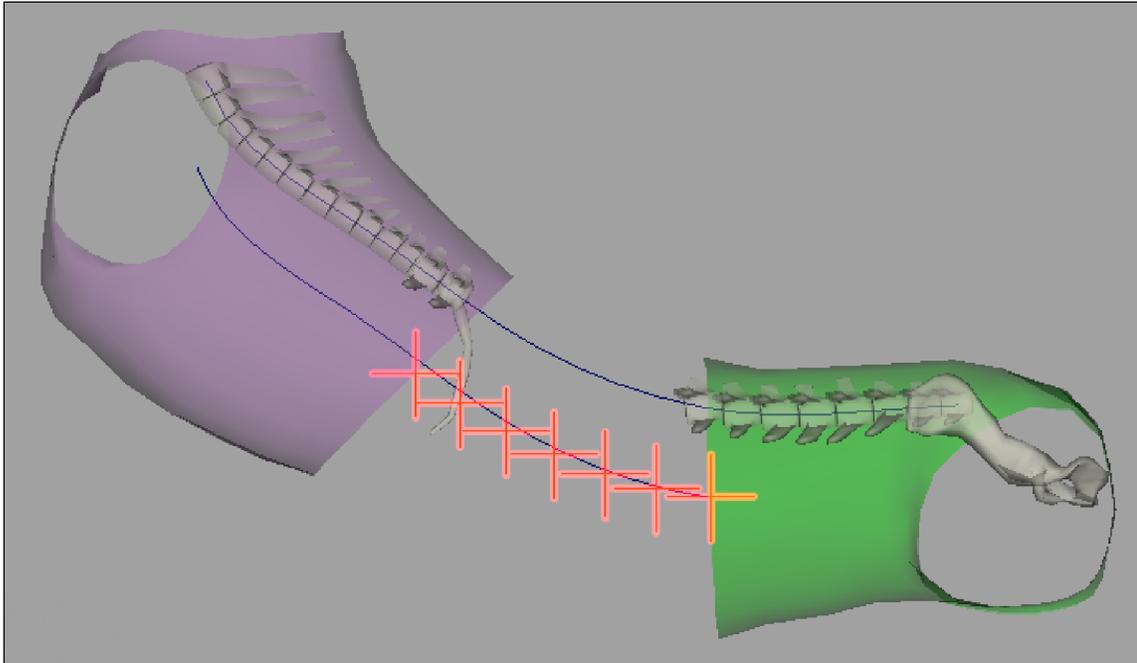


Figure 27. Locator Placement to Define Connecting Extrusion

This process determines the center point position of each edge loop, but does not account for the orientation of the edge loop. In order to determine this orientation, the normal of the plane that approximately describes the out and in borders are compared. This normal is calculated by iterating through each three point combination for all vertices that make up the border, calculating the normal defined by the points at each combination. These normals are then averaged in order to get the normal of the plane. As mentioned, this is only an approximation and is prone to error if an uneven distribution of points skews the results. However, this is deemed an acceptable risk, as border edges will generally be largely planar, so the average normal should largely be

the same no matter the distribution. After calculation, normals are verified to be pointing in the approximate direction of the connection and flipped if not.

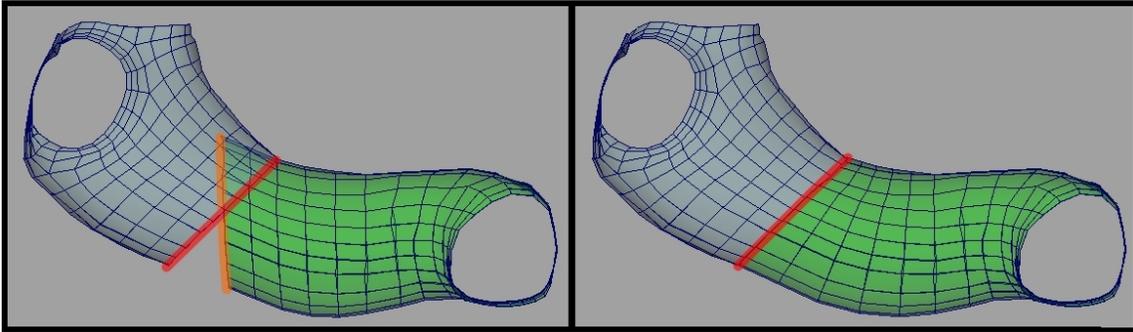


Figure 28. Reorienting Extrusion Border Based on Matching Border Normals

These normals are then used to determine the rotation amount to align the out border normal to the in border normal (Figure 28). It is desired that this angle be signed, so the rotation axis must also be found. This is calculated as the cross product of the out border normal and the in border normal. These three vectors are normalized and the desired rotation amount is as:

$$\begin{aligned}
 x &= \mathbf{v}_1 \cdot \mathbf{v}_2 \\
 \mathbf{y} &= \mathbf{a} \cdot (\mathbf{v}_1 \times \mathbf{v}_2) \\
 \text{angle} &= 2 \arctan \frac{y}{\sqrt{x^2 + y^2} + x}
 \end{aligned}$$

where \mathbf{v}_1 is the out border normal, \mathbf{v}_2 is the in border normal, and \mathbf{a} is the axis around which rotation should be applied.

This yields a signed angle in radians, so the results are converted to angle units

before returning. The returned angle is equivalent to the total amount of rotation to apply to the out border in order to align its normal with the in border. This rotation will be distributed smoothly over the number of segments the extrusion is to be divided into, so the rotation is similarly divided. The previously mentioned offset between in border center point and where the trimmed curve intersects the in border plane is also divided to be applied smoothly over the range of extrusions.

With the additional information of rotational axis and offset, the extrusion locators are further prepared for the extrusion. This preparation occurs by iterating over all extrusion locators. At each step, the locator is first moved relatively by the offset multiplied by the current iteration index. In addition, a temporary locator is created and moved to the current extrusion locator's position and then offset by the border rotation axis. This temporary locator is then used as a target to aim the current extrusion locator's Y axis at. In this way, rotation values applied in Y will occur around the axis the rotation angle was originally calculated on.

Once the extrusion locators are fully prepared, the extrusion process begins. This process occurs as an iteration over the number of expected extrusion divisions. At each step, the border edge is moved through the use of a cluster. The cluster is positioned by parent constraint to the extrusion locator corresponding to the current step number. This locator will be centered in the middle of the current position of the out border vertices. Out border vertices are then added to the cluster's object set, allowing the cluster to deform the vertices. The current extrusion locator's position is then snapped to the next extrusion locator. Because the cluster is still constrained to the current extrusion locator,

it inherits this positional change.

The locator is then rotated by the border rotation amount calculated earlier. The border rotation amount is applied as is, rather than multiplied by the current iteration number because the out border vertex positions are retaining previous rotations and so the rotation amount is accumulating. After the rotation is applied, the out border vertices are correctly positioned for this step. With the border in position for this step, the polySplit action is performed to split the edge ring and create a new edge loop simulating an extrusion.

4.5.3 Alignment of Borders Between Parts

Following skin deformation, the out and in borders will be closely aligned, but likely not coincident, as desired. In order to achieve this coincidence, border points must be snapped together. Because parts can be from a range of different animals and parts can be oriented arbitrarily, it would be impossible to hard-code an exact relationship between each index pair that should snap together on the out and in parts, and therefore requires a procedural solution. Despite the procedural nature of this border snapping, there is still a limitation that in and out parts must share the same vertex count on their matching borders for the snapping to be successful.

Based on the previous deformations that have occurred, the borders should be fairly close to a matching state already. Taking advantage of this assumption, the vertex pair that most closely match position in their current state is assumed to be a matching pair (Figure 29). This pair will be the start of a pair of ordered lists that loop around the border, where each index in the two list represents a pair of vertices that should be

coincident. Although the starting point of the list has been found, it is still necessary to determine the direction both loops should travel such that vertex pairs are correct. In order to find the direction, a second vertex pair needs to be found.

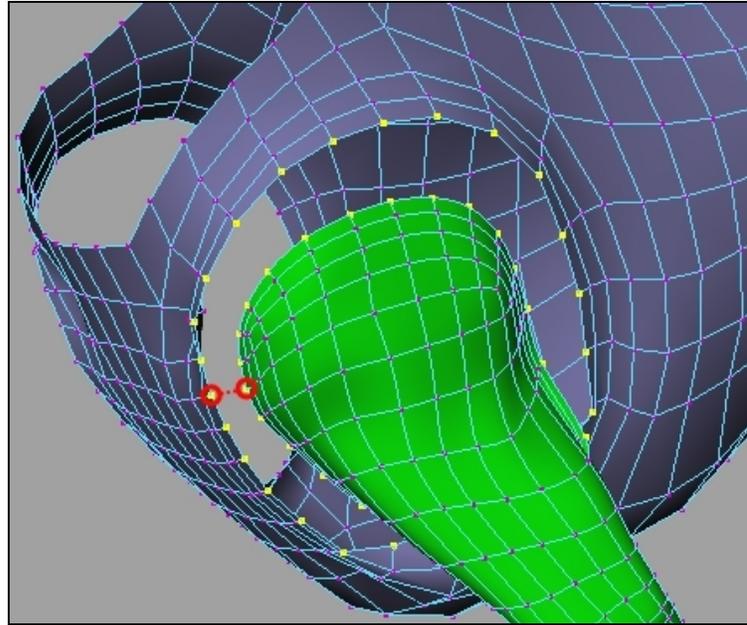


Figure 29. Starting Point of Border Matching Based on the Closest Pair

The normalized vector running from the starting out vertex to an arbitrarily selected neighbor vertex is calculated as the direction to traverse the out loop. On the in border, the normalized vectors for starting vertex to both neighbor vertices are found. These two vectors are compared to the out direction by calculating the dot product between the two. The edge with the greatest dot product is considered to be running in a similar direction as the out vector and the corresponding in vertex is chosen as a pair to the arbitrarily selected out vertex. With the direction to traverse the loop established and

the first two vertices for each loop recorded, it is now just a matter of continuing the traversal around the border edge loops and finding the index if the vertex pairs at each step (Figure 30).

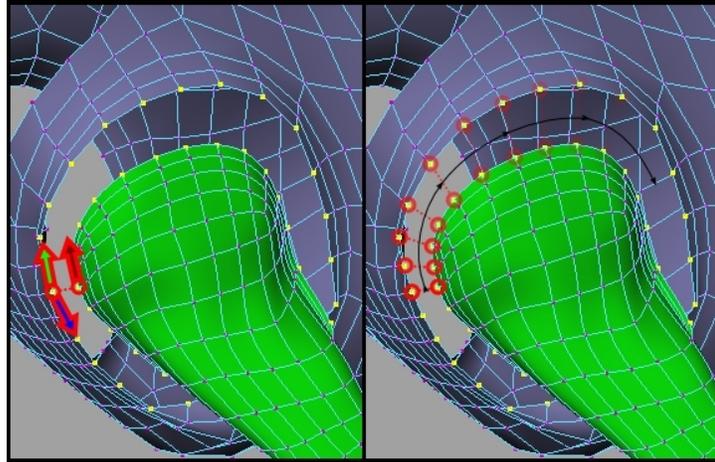


Figure 30. Matching Border Positions Based on Ordered Vertex Pairs

For each pair, the worldspace position of the vertex on the in border is evaluated and the corresponding out border vertex is moved to that position. At each step, a key/value pair is added to a dictionary of pairs, where the key is the out border vertex and the value is the in border vertex. Once all points have been moved to be coincident, this dictionary is returned, to be later used to smooth across parts.

4.5.4 Deformation Smoothing

Deformation bindings for skin vertices are calculated at deformation time by closest point to the bone affector. Furthermore, these bindings are rigid, allowing only a single point on the affector to influence the results. These factors can yield some rough

results as neighboring vertices can, at times, bind to vastly different points on the affector. As such, it is important to perform a smoothing pass following the deformation (Figure 31). Due to necessities such as the need to smooth across two meshes as though it were a single connected mesh, maintain certain vertex positions in the smoothing process, and smooth with a falloff, it was determined that creating a custom smoothing solution rather than utilizing the built-in Maya smooth operation was the correct course of action.

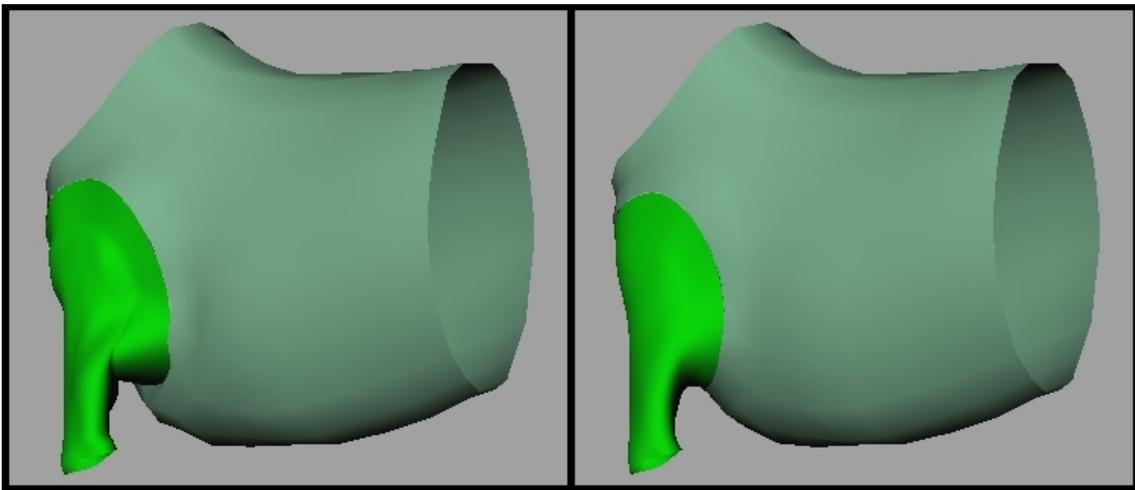


Figure 31. Example Before and After Smoothing of an Uparm-to-Torso Connection

This smoothing solution begins by converting out and in skin meshes to `bmModels`. Smoothing is first processed on the out mesh, which has undergone the bulk, if not all, of the deformation. In this case, smoothing occurs on all vertices of the mesh. Following out mesh smoothing, the in mesh is smoothed, which only smooths vertices within a certain falloff (Figure 32). This falloff occurs between the calculated

maximum distance from the border center point and twice the maximum distance. Any vertices that are less than the maximum distance from the border center get full smoothing applied.

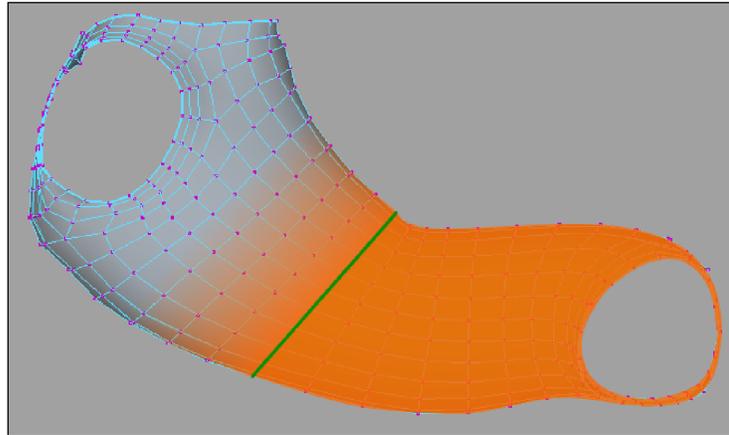


Figure 32. Smoothing Falloff Between Parts

The smoothing itself is enacted by calling the `smoothMesh` function. The `smoothMesh` function begins by making a copy of the model the input `bmModel` was generated from. This copy will later be used as a reference for volume preservation, and must therefore represent the model before any smooth operations are applied. Additional preparation occurs in acquiring a list of vertices that should not be smoothed. Vertices that will not be smoothed are defined as all borders except the one that is currently connecting, as read from the skin attribute file. By not smoothing these borders, it is ensured that other parts that have previously been connected to the part currently being smooth need not be considered as part of this process.

The smoothing process then begins in earnest, iterating over all vertices in the mesh. For each vertex iterated over, the step first checks to see if the vertex is contained

in the list of vertices that should not be smoothed. If the vertex is in the list, the iteration moves onto the next vertex without altering the position of the current vertex. If it is in the list and the smoothing process is set to smooth within a falloff, the step will then measure the distance from the current vertex to the border center. Otherwise, the distance remains at the value of zero where it was set prior to the iteration. The step then checks if this distance is less than the maximum distance in which smoothing is to be applied. Maximum distance is defined as twice input envelope size. If smoothing is to be applied to all vertices, then a distance of zero is compared to a maximum distance of two, and is therefore always true. However, if smoothing is applied by falloff and the distance is greater than the maximum distance, then the step now continues onto the next vertex without altering the position of the current one.

If the vertex does meet the qualifications to undergo smoothing then the vertex is smoothed based on the average position of its neighboring points, extending out two steps (Figure 33). At each step to acquire neighboring vertices, each currently selected vertex is checked against the list of border pairs (acquired previously during border snapping), in order to determine if it has a corresponding vertex on the other part. If this is the case, that corresponding vertex is also selected. This ensures that vertices that lie close to the border can consider points on the opposite part in determining their smoothed position. With all neighbors found, the smoothed position can now be found by calculating the average of the neighbor's positions.

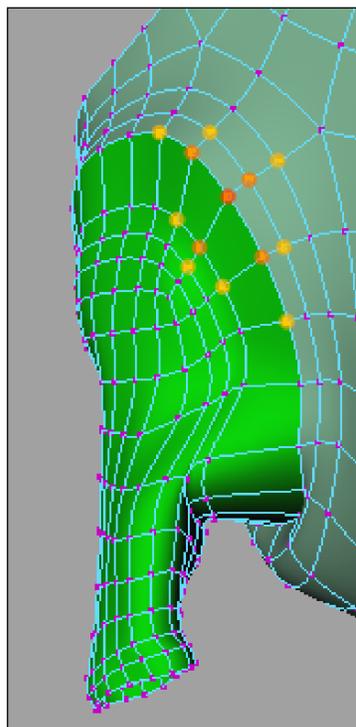


Figure 33. Two-Step Neighboring Vertex Positions Considered for Determining Smoothed Position of Center Vertex

After finding the average position, the offset from original position to average position is calculated as a vector. A multiplier is then applied to this vector in order to apply any falloff. If the distance from the point to the border center point is less than the input envelope size (or all vertices are to be smoothed), then this multiplier remains set to one. If, however, the distance lies between envelope size and maximum distance, the multiplier is set to one minus the interpolation value that describes the current distance's position between envelope size and maximum distance. This creates a linear falloff that can be applied to the offset. It would be a simple matter to apply a more complex falloff here, but linear falloff provided reasonable results. After applying falloff to the offset, it

is reapplied to the original position in order to get the smoothed position. This position is then set as the vertex's position in the `bmModel` and the smooth iteration continues to the next vertex. Once all vertices are complete, the smoothed vertex positions stored in the `bmModel` are transferred to the actual skin mesh.

Because the average of these points is generally occurring over a curved surface, there is a tendency for the results of the averaged points to cause volume loss in the mesh. This is further exacerbated by the fact that, in this implementation, all neighboring points are weighted equally, rather than by surface distance or some other weighting factor. Rather than implementing a more complicated smoothing algorithm, volume loss is compensated by applying an offset found by comparing the current smoothed mesh with the pre-smoothed mesh, duplicated at the start of the function.

Specifically, each vertex in the smoothed `bmModel` is iterated to find the appropriate offset. This offset is calculated based on the current smoothed vertex position to a point where a ray traveling along the vertex normal, cast from the position of the current vertex, intersects with the pre-smoothed mesh. To find this intersection point, four vector positions on a plane are calculated, defined by the current smoothed vertex position the current vertex normal, each surrounding the input point and 90 degrees apart. These four points are generated a very small distance away from the cast point and are used to compensate for bad returns that result when the ray casting operation, `cMuscleRayIntersect`, hits certain edge and corner cases.

A ray is then cast from each of these surrounding points along the same normal. It is the case that if `cMuscleRayIntersect` does not find an intersection it returns origin as

the result. While a better indication of failure would be ideal, this is not the case. Therefore, before accepting the results of the ray cast, the distance of the return point from the line defined by the input point and the normal vector must be checked against. If the distance is within a close enough tolerance to zero, the results are deemed valid. Otherwise, a return value of None is provided, to better indicate the failure state. It should be noted that this implementation does allow for a false positive if the line runs through the origin but `cMuscleRayIntersect` has failed. It is assumed that this is a sufficiently rare case that it need not be accounted for.

The list of ray intersection results is then iterated through to determine their distance from the surrounding points they originated from. This is calculated by getting the length of the vector from cast point to intersection point. These distances are then added to get the average cast distance. The normalized current vector normal is then multiplied by this average distance to yield the offset vector from the current smoothed mesh point to this intersection point, which is saved to a list. Should the ray intersection fail to find an intersection point, the offset vector for this point is set to $(0,0,0)$. This offset vector represents how each vertex should inflate outwards in order to restore the original volume (Figure 34).

Once all vertices on the smoothed mesh have been calculated, the list of vertices are iterated through a second time. On this iteration, the offsets are smoothed by their neighbors, to ensure the rough nature of the pre-smoothed mesh is not reintroduced. Because neighboring indices were saved from the previous smoothing process, it is a simple process to retrieve the neighboring offsets from the list and calculate their

average. Once the average offset has been calculated for each vertex, it is applied to the current smoothed position in order to get a volume-restored, smooth position. When this process has been applied to all vertices, the skin mesh vertex positions are once again synchronized with those stored in the bmModel.

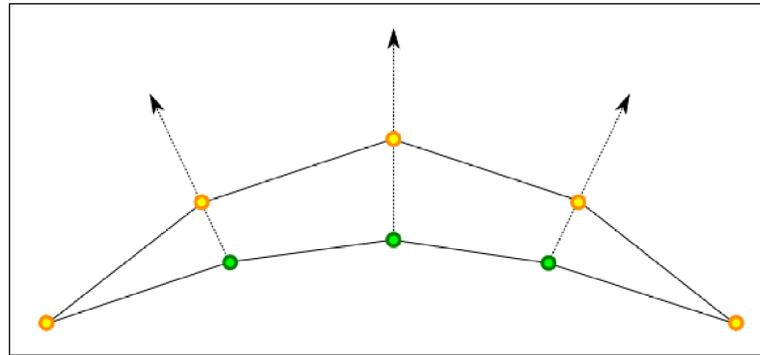


Figure 34. Smoothed Green Points Regain Their Volume by Ray-Casting Against the Original Surface and Finding the Point They Intersect (Represented by Orange Points)

After this process is applied to both the out and in models, it is generally the case that the volume preservation step in the smoothMesh function will often result in the matching out/in border pairs being slightly different in position. It is desired that these pairs remain colocated, so the average of each pair of positions are calculated and the pairs are both set to this new average position. This results in two separate meshes that have been smoothed together such that they now appear to be a single continuous mesh. Upon completion of smoothing, the part connection process is complete.

4.6 Disconnecting Parts

During the course of modifying a creature, the user will likely wish to disconnect parts in order to achieve a different look as they go through the design process. To this end, the beastMaster script currently supports three different methods of disconnecting a part, to give the user the greatest amount of artistic freedom. These disconnection types includes a disconnection that maintains existing deformations, a disconnection that removes deformations but maintains user-applied transformations, and a total reset to default state (Figure 35).

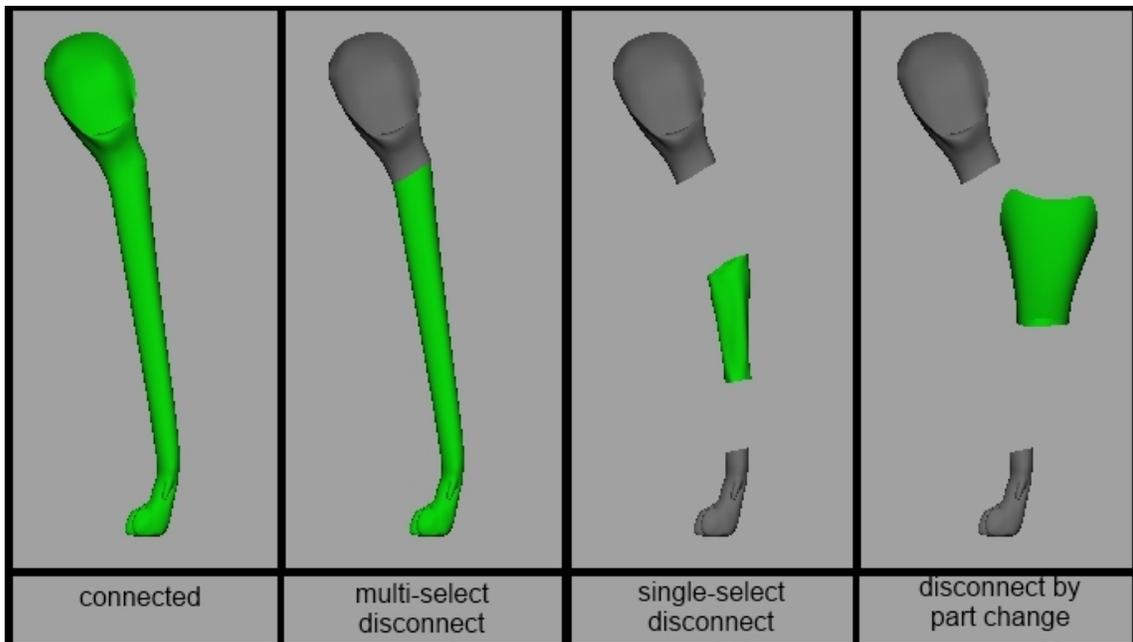


Figure 35. Disconnection Types as Performed on a Deformed Leg

The first disconnection type maintains existing deformations. This means that any skin and bone changes that have already been applied are preserved in the part. The

user enters into this disconnection mode by selecting the disconnect radio button in the beastMaster UI and then multi-selecting a number of parts. The first part selected indicates the part to be disconnected, while additional selected parts are those that the first part should be disconnected from.

The results of this disconnection type only disconnects the parts in their connection lists, maintained in the partGroup node that has previously been created in the scene. These connection lists are responsible for associating parts with other parts they have been connected to, in order to give the user the illusion that it is a single part when applying further manipulations. By removing this relationship, the parts continue to look as though they are connected, but can once again be transformed individually by the user. This allows the user to remake connections from its deformed state.

The disconnection applied in the connection list is enacted through the remFromConnList function in the bmDuplicatePart module. This function begins by finding all partGroup nodes in the scene. In practice there should only be a single partGroup generated in a scene, but the function supports the possibility of garbage partGroups without causing errors.

For each partGroup node, the list of connected parts is iterated through and paired with the part to be disconnected. Then the connection lists for each of the items in this pair are retrieved from the partGroups node in the form of a comma-separated string list. The opposite item in the pair is then found and removed from the string list and the list is reapplied to the attribute. If the resultant string list is found to be empty after the removal of the connection, the attribute is instead deleted. If, after iterating

through all connections, the partGroup node is empty, the node itself is deleted. Once this is complete, this type of disconnection process is complete.

The second disconnection type removes the deformations applied to the part but maintains transformations applied by the user. This allows the user to revert changes applied to the part through the connection process, without moving the part back from default position to its desired location. The user enacts this disconnection mode by selecting a single part, rather than a multi-selection of parts, and clicking the disconnect button.

It should be noted that while the disconnecting part is reset to its original position, other parts that have connected to it will remain deformed. This is due to the fact that the part being disconnected from the still-deformed part is potentially not the only connection that has resulted in a deformation of this part. Because joint structures are generated and deleted at each connection, it is impossible to differentiate between deformations caused by the disconnecting part and deformations caused by other connected parts. It would therefore require that connections be rebuilt from the beginning or the state saved at each deformation in order to support the removal of a single connections deformations. Either method would be a burden on the system under the current implementation. Should the user wish to remove the deformations from parts surrounding the disconnecting part, the workflow would be to perform a disconnect of this type on the surrounding parts and then reapply the desired connections.

This disconnection type is implemented by calling the importPart function. This function first deselects the selected part in order to bake any transformations still

existing on the control node into the part group. The part group will then contain any transformations applied by the user onto the part. The part group's xform is then queried and stored, after which the entire part is deleted and reimported in its default state. The previously acquired xform is then reapplied to the part, now in its default position, in order to return it to the transformation previously set by the user.

This preservation of user-set transformations can also be utilized when switching between parts of different animals. It is for this reason that this disconnect type is implemented in the importPart function. This allows the user to switch between different animal types for a part without needing to move the part back into place with each change. While different animal parts will appear in approximately the same location upon switching, there will be some positional change as the xform is applied around the part's pivot point, which will vary between parts of different animals.

The final disconnection type, which removes deformations and transformations from the part, is enacted through the part selections in the user interface rather than the disconnect button. By switching the part to a value of None and then returning to the current animal, the user can return the part to its original state in its default position. This is also enacted through the importPart function, as described above. The difference is that the part will first be deleted when the None option is selected. Then, when selecting the desired part again, there will be no xform with user transformations stored to reapply to the part. This results in the part getting imported anew with no transformations applied.

5. CONCLUSIONS

The system created in this thesis strove to create a tool that would aid in the design of realistic creatures, utilizing established rules based on anatomical study. The rules established for use in this system, although perhaps too generalized, are of value and can be deemed successful. The implementation of the system can be regarded as broadly successful by the fact that a user can, in fact, use it to automatically combine parts of different animals, through the established rules, into a new, chimeric creature. The resultant skin mesh does work as a base mesh for further embellishment through modeling or as a silhouette for draw-over. The resultant skeleton does make for a good sculpting or rigging reference.

However, with regards to actually being a useful tool, the implementation fails. Foremost is the fact that the feature set called for, in order to make it truly useful, was far more complex to implement than originally thought. As a result, many of these features were dropped in the interest of keeping the work within the scope of a thesis topic. As mentioned in the Future Works section, this system would need to rely on more than just skeletal connections to give the truly impressive, scientific results originally envisioned. The current set of implemented rules governing skin deformations are simply too few to provide good, strong, scientifically guided results.

Converse to the need for additional complexity within the system is the fact that the current level of complexity results in a connection process that is likely too slow to be effectively used as a design tool. Efforts at optimization did little to speed up the

process. This failing is likely due to the fact that this process was implemented as a script, rather than implementing it through compiled code, utilizing the program's API. This was originally a decision meant to allow this system to be easily ported to other programs outside of Maya, due to its Python implementation. However, because so much of the Python implementation relies on Maya-specific commands, the possibility of a simple conversion seems suspect.

The original vision of what this system would be, using thoroughly modeled systems of underlying anatomy to create highly realistic creatures based on scientific knowledge of how these systems commonly connect between disparate creatures was perhaps too lofty for the scope of this thesis. While this thesis did not succeed in reaching this grand vision, it did make some good forays into the effort, which could perhaps be picked up by another intrepid researcher. Considering this, along with the scope of the work, the successes likely outweigh the failures in this work.

6. FUTURE WORK

The scope of this thesis leaves a great many possibilities open for future work that could be done to enhance the tool's usefulness, both from the standpoint of creature design and from the standpoint of scientific accuracy. From an artistic standpoint, the most obvious addition would be the expansion of the library of animals available to the designer for use in the modular construction of fantasy creatures. A limited library was created for this thesis as a proof of concept, so it will be necessary for future work to be done in building this library to ensure the tool is of true use. As the system includes setup tools to facilitate the creation of additional animals that can be used with the system, these can be created by anyone with even a cursory knowledge of how the system functions.

Another way to expand the options available to the designer would be to expand the scope of the thesis beyond the tetrapod superclass. This would allow the designer to make use of the parts of creatures such as insects and fish; creatures whose parts are often referenced in the design of fantasy creatures. Because these animals have a very different or complete lack of skeletal structure, these creatures would require substantial work to integrate into the system.

It could also be beneficial to introduce extinct animals, such as dinosaurs, to the available pool of animals. This work steered away from extinct animals due to the more limited knowledge regarding their skeletal structure. However, more concentrated research into this area could allow them to be added to the parts library.

It might also be of use to the designer to allow for the representation of fur, feathers, scales, and the like. These elements often play an important role in defining a creature's form or silhouette. Similarly, it would be useful to the artist to allow a greater selection of appendages that can vary, even within a species, such as horns, antlers, and claws. These too can drastically alter a creature's form, but represent a greater specificity in the subdivision of the creature into component parts than the current system allows.

From the standpoint of scientific accuracy, adding more underlying anatomy than just the skeletal system in influencing the exterior skin would be highly beneficial. This was one of the original goals of this thesis that proved too ambitious. However, future work towards implementing muscles, tendons, organs would greatly increase the scientific accuracy of the connections and aid in designing creatures with even greater realism.

Although this system does introduce a rudimentary concept of allometric scaling at connection time, an even greater use of concepts from this field of study could be of use. For instance, in this system, bones currently scale under the same generic proportionality. These changes in bone scale directly affect the skin. In reality, there are allometric proportions between skin weight and skeletal weight that could be applied to get the proper skin change as the bone scale changes. Furthermore, all bones would not realistically scale equally. Depending on their function, support loads, stance, and other factors including environmental variables and season, these proportions could change dramatically from bone to bone. Due to the great number of factors that would have to

be considered for a more fleshed out implementation of allometric scaling this would perhaps be suitable for a thesis topic all its own.

Environmental variables influence more in the evolution of an animal than just its scaling. As such, it might be of use to allow the designer to input some environmental variables such as gravity, temperature, climate, habitat, and available food that could guide the artist in a realistic direction. For the purpose of further work on this thesis, it would be unwise to allow these variables to dictate the design of the creature, interfering with artistic freedom. However, some indicator as to the plausibility of the design currently enacted (some sort of anatomical plausibility scale) could be of great value to further enhance the believability of the creature by allowing it to best fit in with its surroundings.

Because the designer is allowed to change the orientation of parts at will, there is the very real possibility that the resultant creature's stance would be unstable in a realistic environment. There is currently no way to prevent this and no way for the designer to know that this is the case. One interesting course of future work could potentially be the implementation of a physical simulation of the creature, that would apply environmental forces such as gravity based on the creatures estimated weight, to see if the creature's bones can handle their current loads and if it can stand given its default stance. This would likely require the aforementioned addition of a muscle structure to properly predict the creatures ability to support itself.

Finally, it might be of benefit to expand the tool to create rigged, animatable characters. Although such characters would likely be of little use for creating “hero”

creatures with specific needs, a simple rig could be of great use for creating background creatures of for use in 3d pre-visualization, allowing for simple, generic movements. By extending setups that will likely already be in place to allow the function of the proposed tool, it may not be too much of a stretch to implement a simple rig following the completion of the creature design process.

REFERENCES

- Biewener, A. 2005. "Biomechanical Consequences of Scaling." In *The Journal of Experimental Biology*. **208**, 1665-1676.
- Bourke, Paul. "Calculating the Area and Centroid of a Polygon." *Notes on Polygons and Meshes*. paulbourke.net Jul. 1988. Web. 3 Mar. 2013.
- Cani, M. and Angelidis, A. 2006. "Towards virtual clay." In *ACM SIGGRAPH 2006 Courses* (Boston, Massachusetts, July 30 - August 03, 2006). SIGGRAPH '06. ACM, New York, NY, 67-83.
- Davis, S. B. and Moar, M. 2005. "The amateur creator." In *Proceedings of the 5th Conference on Creativity & Cognition* (London, United Kingdom, April 12 - 15, 2005). C&C '05. ACM, New York, NY, 158-165.
- Hildebrand, M. and Goslow, G. *Analysis of Vertebrate Structure, 5th Edition*. New York, NY: John Wiley and Sons, Mar 1998. 29, 434.
- Huante, Carlos, [The Techniques of Carlos Huante Volume 1: Creature Sketching and Design](#). DVD. The Gnomon Workshop and Design Studio Press, 2006
- McLaughlin, T. 2005. "Taxonomy of digital creatures: interpreting character designs as computer graphics techniques." In *ACM SIGGRAPH 2005 Courses* (Los Angeles, California, July 31 - August 04, 2005). J. Fujii, Ed. SIGGRAPH '05. ACM, New York, NY, 1.
- McLaughlin, T. 2006. "Taxonomy of digital creatures: defining character development techniques based upon scope of use." In *ACM SIGGRAPH 2006 Courses* (Boston, Massachusetts, July 30 - August 03, 2006). SIGGRAPH '06. ACM, New York, NY, 1.
- McLaughlin, T. and Sumida, S. S. 2007. "The morphology of digital creatures." In *ACM SIGGRAPH 2007 Courses* (San Diego, California, August 05 - 09, 2007). SIGGRAPH '07. ACM, New York, NY, 1.
- Page, Neville, [The Techniques of Neville Page Volume 1: Character Design - Fantasy Wildebeest](#). DVD. The Gnomon Workshop and Design Studio Press, 2004
- Scheepers, F., Parent, R. E., Carlson, W. E., and May, S. F. 1997. "Anatomy-based modeling of the human musculature." In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (Los Angeles, California, August 03 - August 08, 1997) SIGGRAPH '97. ACM Press/Addison-Wesley Publishing Co., New York, NY, 163-172.

Schmidt-Nielson, K. *Scaling: Why Is Animal Size So Important?*. Cambridge, United Kingdom: Cambridge University Press, 1984. 7, 8, 32

Sims, K. 1994. "Evolving virtual creatures." In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques* (Orlando, Florida, July 24 – July 29, 1994) SIGGRAPH '94. ACM, New York, NY, 15-22.

Whitlatch, Terryl. "The Creature Design Method." *Tales of Amalthea*. talesofamalthea.com 2013. Web. 2 June 2013.

Wilhelms, J. and Van Gelder, A. 1997. "Anatomically based modeling." In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (Los Angeles, California, August 03 – August 08, 1997) SIGGRAPH '97. ACM Press/Addison-Wesley Publishing Co., New York, NY, 173-180.

APPENDIX A

Table A-1: Major Anatomical Parts Table A-2: Connection Types

Part Name	Connection Type
1. Head	
2. Neck	
3. Chest	
4. Abdomen	
5. Tail	
6. RtUpperArm	
7. LtUpperArm	
8. RtForearm	
9. LtForearm	
10. RtManus	
11. LtManus	
12. RtThigh	
13. LtThigh	
14. RtLowerLeg	
15. LtLowerLeg	
16. RtPes	
17. LtPes	
	Type 1 Limb Bones (<i>ex. Femur</i>)
	Type 2 Scapula -> Humerus connection
	Type 3 Vertebra -> Vertebra/Head
	Type 4 Head -> Neck Vertebra
	Type 5 Ribs -> Chest Vertebra
	Type 6 Ribs -> Scapula
	Type 7 Vertebra -> Scapula/Pelvis
	Type 8 Pelvis -> Thigh
	Type 9 Pelvis/Sternum -> Abdomen Vertebra

APPENDIX B

FILE TYPES

For any animal installed for use with the beastMaster tool, there are four types of files that are required for system to function properly when working with parts of this animal. These files must be installed in specific directories under the installed animal directory. These files are generated through the Setup tools as each new animal is created for use with the system. The four types are defined as Part Files, Bone Connection List Files, Skin Attribute Files, and Bone Attribute Files.

Part Files

Part files are Maya scene files (.ma) that represent one of the Major Anatomical Parts of an animal. These files contain a properly structured group of skin and bone meshes, as well as any required connection and alignment locators. These files are imported into Maya when the user selects the associated animal in the beastMaster UI's Part Selector.

Bone Connection List File

Bone Connection List files define the series of bone connections that must occur when one part is connected to another. A Bone Connection List file corresponds to each Major Anatomical Part, specific to each animal, as the exact bones present in each animal may vary. The data stored in the file consists of an ordered list of bone-to-bone connections for each part the file's corresponding part can connect to. While bone-to-bone connections are generally all listed explicitly, in cases where there is a numbered

series of bone connections, such as a chain or rib or vertebra connections, a single generalized connection will be listed. At run-time, this generalized connection will be recognized and expanded to a full list, in order to keep the bone-to-bone connection lists limited to a reasonable size.

Skin Attribute Files

Skin Attribute Files define the indices of the edges that define the borders of the corresponding part. Skin Attribute files correspond to each Major Anatomical part and are unique to their associated animal, due to the fact that skin meshes vary between animals. However, it is essential that for each animal, files relating to the same part have the same number of edges listed. In addition to defining border edge indices, each border is paired with another, indicating its opposite border to the system.

Bone Attribute Files

Bone Attribute Files contain the information necessary to procedurally create the joint structure used to deform the bones at connection time. For each animal, there is a Bone Attribute File corresponding to every bone in the skeletal structure. The Bone Attribute File also contains data describing, for each possible bone the files corresponding bone can connect to, the prescribed connection type, the direction the joint structure should be built in, and which bone should be duplicated, in the case of vertebra.