THE STRETCH-ENGINE

A METHOD FOR CREATING EXAGGERATION IN ANIMATION THROUGH SQUASH AND STRETCH

A Thesis

by

ZAID HAMED IBRAHIM

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

MASTER OF SCIENCE

Chair of Committee,	Tim McLaughlin
Committee Members,	Frederic Parke
	John Keyser
Head of Department,	Tim McLaughlin

August 2018

Major Subject: Visualization

ABSTRACT

Animators exaggerate character motion to emphasize personality and actions. Exaggeration is expressed by pushing a character's pose, changing the action's timing, or by changing a character's form. This last method, referred to as squash and stretch, creates the most noticeable change in exaggeration. This work introduces a prototype tool, the Stretch-Engine, to create exaggeration in motion by focusing solely on squash and stretch to control changes in a character's form. It does this by displaying a limbs' path of motion and altering the shape of that path to create a change in the limb's form. This paper provides information on tools that exist to create animation and exaggeration, then discusses the functionality and effectiveness of these tools and how they influenced the design of the Stretch-Engine. This method is then evaluated by comparing animations of realistic motion to versions created with the Stretch-Engine. These stretched versions displayed exaggerated results for their realistic counterparts, creating similar effects to Looney Tunes animation. This method fits within the animator's workflow and helps new artists visualize and control squash and stretch to create exaggeration.

DEDICATION

This thesis is dedicated to my wife Monica, who stood by me, and my family, who always encouraged me. Thank you all for believing in me and supporting me as I work towards my dream.

ACKNOWLEGEMENTS

This project could not have been achieved without the incredible support of the faculty and staff at Texas A&M's Visualization department. I would personally like to thank my committee chair, Prof. Tim McLaughlin, who guided and supported me as I developed the ideas for this project. Without his help, this project would have never been completed. I would also like to thank my other committee members, Dr. Frederic Parke, and Dr. John Keyser for their advice and counsel.

I would like to also thank my friends, colleagues, and family for their support throughout my academic career at Texas A&M. Specifically Joshua Seal, Soumitra Goswami, Rushil Kekre and Nicholas Forasiepi. Your help and support were invaluable and I could not have succeeded without it.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of chair Professor Tim McLaughlin and Dr. Frederic Parke of the Department of Visualization and Dr. John Keyser of the Department of Computer Science.

All work for the thesis project was completed independently by the student, with edits made to the paper by Dr. Frederic Parke of the Department of Visualization and Monica Parshley and Alya Ibrahim and Texas A&M's University Writing Center.

Funding Sources

There are no outside funding contributions to acknowledge related to the research and compilation of this document.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION[1]

Animators rely on an understanding of the principles of animation [1] to create the motion and emotion in their work. These principles were developed by the animators at Disney Animation and became a framework for what to do when practicing animation. By keeping these principles in mind, artists could create appealing motion and develop their own style of work. However, determining whether or not a certain principle was important in the creation of their artistic style would be difficult without proper training. This paper chooses to discuss the importance of one principle in particular, squash and stretch, described as "One of the most important principles of animation" in the book *The Illusion of Life* by Frank Thomas and Ollie Johnston.

To better understand the importance of squash and stretch, a brief overview of existing tools that focus on this principle is discussed. Some of these tools focus on using simulations to control animation and the degree of squash and stretch. The final result is an automatically created motion with little or no manual input from the user. These tools may use realistic physics in the calculations that affect squash and stretch, or they may use other techniques to create cartoon-like effects. The other techniques include tools that allow the user to visualize the changes squash and stretch can have on the overall animation and provide the user with controls to affect these changes. These tools produce exaggerated animation results, but they all have shortcomings, all of which were considered when developing this method.

[1] Parts of this section are reprinted with permission from "The stretch-engine: a method for adjusting the exaggeration of bipedal characters through squash and stretch" by Zaid H Ibrahim, 2017. In Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '17), Stephen N. Spencer (Ed.). ACM, New York, NY, USA, Article 30, 2 pages. DOI: https://doi.org/10.1145/3099564.3106639

1.1 Purpose

The purpose of this paper is to describe a method to create exaggeration in realistic motion by controlling changes in a character's form through squash and stretch. The *Stretch-Engine* is a tool developed to achieve this goal by generating a 3D motion curve that visualizes a limbs path of motion and changes the shape of that path to impose squash and stretch on the character's limb. The Stretch-Engine also contains a procedurally generated bipedal rig that has the ability to squash and stretch its limbs as well as a graphic user interface (GUI) to create 3D motion paths and control the form of these curves. The code used to build the Stretch-Engine is written in Python and is integrated into the existing Maya animation software.

1.2 Significance

The Stretch-Engine allows animators to control changes in a character's form by changing the degree of squash and stretch through altering a limb's path of motion. With access to more control in squash and stretch, animators are able to develop better quality exaggerated motion. The result is an efficient way of controlling squash and stretch during animation of bipedal characters. The Stretch-Engine is demonstrated using three animations based on realistic motion that employs squash and stretch to create exaggerated versions.

Figure 1: Example sketch of limb exaggeration through squash and stretch based on Looney Tunes episode "To Duck or Not to Duck." Episode released March 6, 1943; Directed by Charles M. Jones.

1.3 Terminology

There are specific terms and phrases used that describe certain processes of development when creating an animation tool. This paper includes some of this nomenclature and uses it to explain the components of the animation tool. For this paper, the terminology used and their definitions are listed below:

A *Rig* is the process and end result of taking a static geometric model, creating an internal digital skeleton, a relationship between the geometry and skeleton, and adding a set of controls that the animator can use to manipulate the character around.

A *Locator* is an object within the Maya interface that visually represents a point in 3D space. It can contain translation, rotation and scale data.

IK, or inverse kinematics, is the mathematical process of determining movements of an object based on the motion of its end effector.

A *Controller*, in the context of this paper, is an end effector for an IK chain within a rig. Allows the animator to interact with and animate their characters.

A *Curve* is a 3D line based on user created keyframes that defines the path of motion that the end effector follows. It is a combination of user input and software generation. The curve is displayed by the software interface.

Geometry is a collection of surfaces organized in a hierarchical structure that forms the character body.

Joints are Maya defined objects where the articulation of the character occurs, the pivot points. Connected to the geometry by weighting functions allowing it to move and deform the character.

Bipedal, a term used to define an animal that walks on two legs.

Right-Hand Coordinate System, or the positive x, y, and z axes, point left, right, and forward, respectively. Positive rotation is counterclockwise about the axis of rotation.

A *Spline* is a smooth curve that passes through or near a set of fit points. Defined with either fit points or control vertices.

Keyframes transform objects or skeletons over time by setting keys, arbitrary markers that specify property values of an object at specific times.

In the terms of this paper, a *Tool* is a set of functions contained within a user interface that provides the user with resources to complete a specific task

2. BACKGROUND AND RELATED WORK

As animation developed, certain expressions were used repeatedly. Such expressions included “aiming,” “overlapping” and “pose to pose.” These expressions suggested that certain animation procedures were becoming specialized and were being given names. These terms were used over and over again, becoming verbs and then nouns. Physical aspects were being used to describe drawing techniques, such as “Look how stretchy that character is.” These terms were ways animators described the success of their work and their peers [1].

As time went on, animators would continue to search for better methods of explaining drawings to one another. These methods were ways to teach others what they have learned and what was successful in producing appealing animations. These methods were techniques they had learned to follow and became a set of guidelines they could quantify and study. Animators used these guidelines as references for new projects they would work on. Although these techniques may not always create the expected result, they allowed for structure and reassurance in the creative process.

As each of these techniques were named, they were analyzed, deconstructed and perfected. They became the foundation for new animators to follow and improve their skills. These techniques later become the *twelve fundamental principles of animation*: Squash and Stretch, Anticipation, Staging, Pose to Pose, Follow Through, Slow In and Slow Out, Arcs, Timing, Exaggeration, Solid Drawing and Appeal. As described in the book *The Illusion of Life*, the most important of these principles is Squash and Stretch [1].

An example of “Squash and stretch” is when an object is stretched as it approaches a collision, squashed as it collides, and then stretched again as it rebounds [3]. The purpose of this principle is to give a sense of weight and flexibility to an object. When an object, like a chair, is moved, there is a rigidity in its form that is emphasized by its movement. Anything composed of flesh, however, shows deformation of its shape when progressing through an action. The “squashed” position of an object depicts its form bunched up and pushed together when a compressive force is applied. The “stretched” position shows the object in an extended position but with a similar shape to its original form. Figure 1 is an example of such a stretch, Daffy Duck’s arm is stretched out to emphasize the punch’s impact.

Animators apply the principles of animation to create their own style of animation. Such styles range from closely following reality, such as the work by Hayao Myazaki, to breaking as many physical constraints as possible, such as “The Amazing World of Gumball” created by Ben Bocquelet. This change in style is most noticeable when using squash and stretch. It can also emphasize the effect exaggeration has on motion. A character’s torso may be slightly compressed when hit in the head or the torso may completely flatten along with the head. The difference is the desired effect the animator wants to produce, or how “cartoony” the character is meant to be. This is the artistic impact squash and stretch has on animation. When studying tools used for animation, there are two different approaches. These can be defined as (1) animation based in physical realism and (2) animation controlled by artistic preference.

2.1 Realistic Approaches

When creating tools for animation, one approach focuses on physics-based simulations to calculate animation realistically. This approach means that all motion created is done by a physics-based process rather than being created manually. As a result, physics-based characters and objects interact in accordance with the laws of physics. Physics-based simulations, which create all forms of motion without the use of motion capture data, have been a topic of interest in the animation industry for quite some time but have not been adopted widely in expressive character animation. Currently, physics-based simulations are time and resource intensive, making manual or motion capture approaches the more efficient option.

There are many techniques that rely on physics-based calculations; one example is the method created by Geijtenbeek [25]. He developed a physics-based simulation to animate bipedal characters with the use of biomechanical constraints. These constraints are based on results from biomechanics research to help create the perception of naturalness. Without the use of these constraints, simulated characters move in ways that are physically valid but seem stiff or robotic. Geijtenbeek's technique created visually appealing walk cycles and performed well for a number of differently shaped bipedal characters. Although this technique creates satisfying animation, it did not take into account form deformations such as squash and stretch.

Hahn [12] developed an early system that simulated the dynamic interaction among rigid bodies. This system took various forces, such as elasticity, friction, mass, and moments of inertia to produce rolling and sliding. This technique helped animators create

realistic results by using equations developed to developed to calculate the aforementioned forces. This method also adjusted the physical calculations to create the desired animation if the realistic values could achieve them. The final result will be an animation that follows realistic or pseudo-realistic motion based on the physical attributes assigned to the object. These pseudo-realistic results refrain from manipulating the object's form and can only be created if the laws of physics are manipulated to create the desired effect.

Mordatch et al. [14] created a trajectory optimization approach to animate human activities that are driven by the lower body. This approach was based on contact-invariant optimization, a technique that smooths out discontinuities in the specified objectives to create a single optimized search for possible motion trajectories. The goal of this project was to automatically create realistic, lower body motion, such as running and jumping. These optimizations evaluated the contact positions and forces of the character's feet to calculate the resulting motion in the rest of the body. The technique used these physical calculations to move the character's feet and, from these movements, calculated motion for the body. Like Geijtenbeek it could not alter the body's geometric form as the calculations relied on that structure to remain constant.

Shapiro and Lee [11] developed an interactive system that helped animators create more physically realistic motion by assuming the character obeys the laws of physics. This method was designed to inform animators of the changes needed to make motions of an animated character physically correct. Their tool used physical characteristics such as center of mass, angular momentum and balance to create an optimal motion path. The system comprises of two tools that were integrated into keyframe-based animation

software. The first is a ballistic path tool that compares the character's path of motion based on the originally created animation to a new path generated by the system. This ballistic path allows an animator to adjust the original motion to create a more physically correct animation. Figure 2 shows an example of an original motion path (blue) compared to the new ballistic path (red) of an animated character. The second tool is an angular momentum tool that rotates a character's global orientation to achieve the desired angular momentum. The degree of rotation is calculated using velocity, inertia, and momentum values based on the character's root, the highest joint in the character's joint hierarchy. The character is rotated without affecting the animation of each body part to preserve the animation style. Their tool allowed the animator to visualize and control changes in a manually created animation and could even adjust exaggerated animations to fit within a live environment. However, it does not include a feature to alter form when working with exaggerated animations.

Chenney et al. [3] simulated squash and stretch for simple bodies, basic geometry, and shapes by using velocity and collision parameters. This approach focused on creating a procedural animation system by using pseudo-realistic physics calculations that simulated object movements through space. By applying the velocity and collision parameters, they were able to calculate how the object's form would change during motion and visualized the resulting squash and stretch. The animator could control an object's motion through a set of parameters that affected features of the motion, such as the degree of squash and stretch. Although this technique was able to produce appealing squash and stretch, it

focused solely on animating simple shapes rather than more complex objects, such as a bipedal character.

Tan et al. [26] created a method of animating soft body characters, which are geometric models that do not contain a skeleton, through volume preservation and muscle contraction calculations. The characters used contained dense topology, the geometric structure that makes up the objects geometry, that allowed the form to shift and bend into any position. They also contained muscles fibers that controlled changes in form by contracting and relaxing. Due to the high topology and large set of muscles fibers that controlled these characters, moving characters by hand could be quite difficult. To manually animate the user would have to modify each fiber individually to create movement in the soft body. Due to this complexity these soft body are animated by setting goals, such as moving a point along the character to a new position, by setting a trajectory for the center of mass, or by regulating the character's linear or angular momentum. These goals are analyzed by a physics solver to determine a change in muscle fiber position. After the new muscle conditions are determined the solver is run again, resulting in a moving soft body. This method, although complicated, produced satisfying soft body character movement. However, this method was designed solely for simple soft body creatures and while it could be used for sections of bipedal characters, such as the tongue, it would not be the optimal controller for animating a full body character.

These physics-based tools were effective in simulating motion under realistic conditions. They provided calculations to create physically correct motion and some tools allowed manipulation of these calculations to create changes in the results. Others showed

representations of changes in motion to compare differences in results; many of the discussed methods could be applied a variety of characters. However, the methods that produced satisfying animations for complex characters did so without deforming the affected objects geometry. Those that could produce squash and stretch were only able to create the effect with simple geometric shapes, or the effect was to create realistic squash and stretch for characters that did not contain a rigid form.

The ability to deform complex geometry is required when creating squash and stretch animation in animated characters. Due to the wide variety of artistic styles that rely on changes in a character's form, a system based on physical realistic constraints may not create the cartoon-like animation desired by the artist. For this reason, the Stretch-Engine focuses on non-realistic approaches to calculate animation which emulates traditional animation techniques.

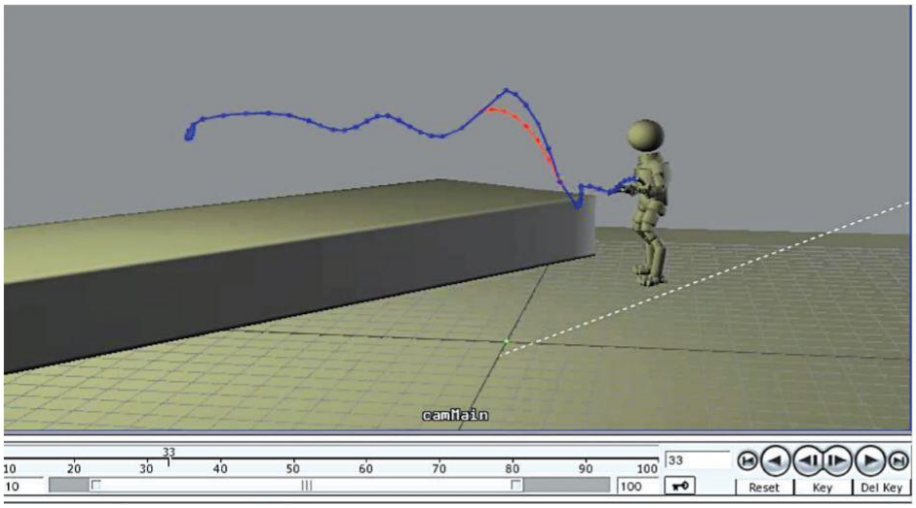

Figure 2: Comparison of the center of mass path created from hand-made animation (blue) and the ballistic path calculation (red). Reprinted from *Practical Character Physics for Animators* [11].

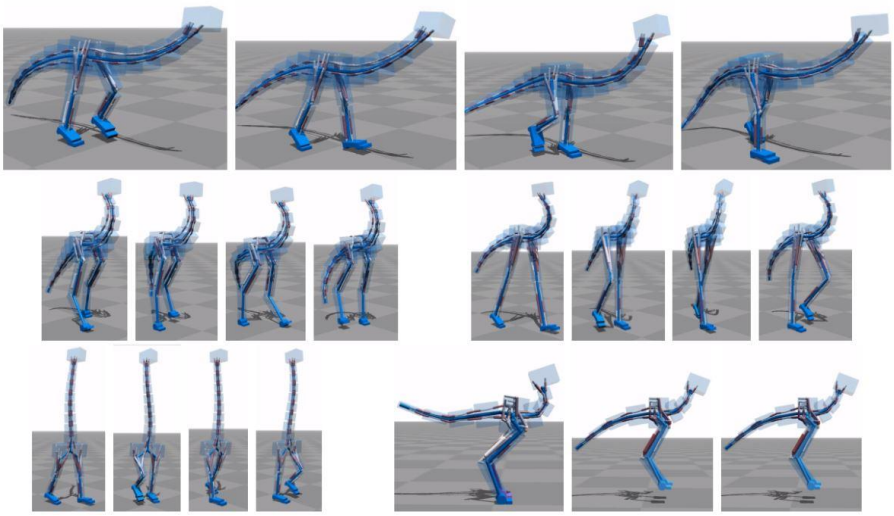

Figure 3: Examples of different walking gaits for a variety of bipedal creatures created through the physics-based simulation. Reprinted from *Animating virtual characters using physics-based simulation* [25].

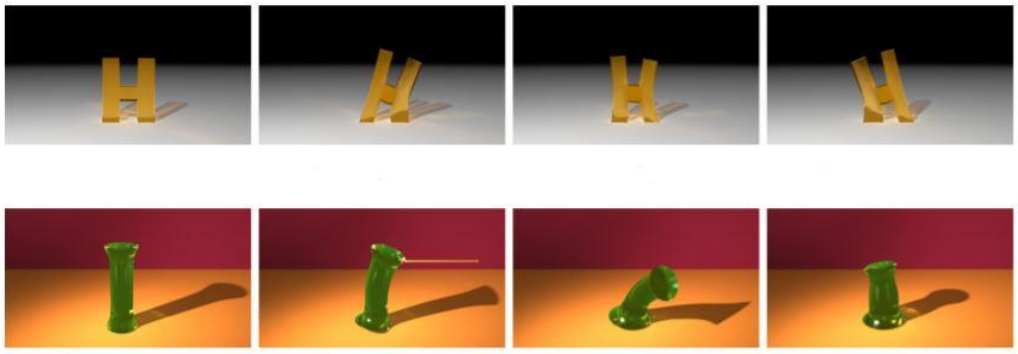

Figure 4: Results of animating two soft body forms. The H swings side to side and the I maintains balance by lowering its center of mass. Reprinted from *Soft body locomotion* [26].

2.2 Non-Realistic Approaches

A non-realistic approach to animation focuses on producing an expected style of animation rather than an animation that follows the laws of physics. Because of this focus on style, many of these approaches allow the character's geometry to deform and create poses that the actual physiology of an object could not achieve. These techniques can also

receive their initial data from a variety of sources. These sources include inputted values for motion tracking data, two-dimensional drawings, video files and procedural calculations.

An example of a non-realistic approach is one created by Roberts and Mallett [5] that resolved squash and stretch using an ‘example based’ technique, in which the resulting deformation was generated from a set of poses. A “pose” was defined using a number of dimensions such as position, velocity, and acceleration. The user adjusted these parameters to create a new pose for the selected object. To create changes in form, the user could iterate through the existing animation and change these parameters to create new poses at different times. The software then applied these poses to the original animation to create the cartoon-like animation. This approach created animations that squash and stretch to exaggerate motion and allowed animators to see changes applied with real-time feedback. However, each time a pose is created a duplicate mesh is placed within the scene. If the animation requires numerous pose changes, the scene file can become heavy and cluttered.

Wang et al. [7] used an inverted Laplacian of Gaussian (LoG) filter to alter inputted realistic motion data to create exaggerated actions. The filter created a smooth and inverted version of the motion data’s acceleration and replaced the original data’s acceleration with the newly created one. The new acceleration altered the character’s movement by having it move further away from the start position and then overshoot the stop position. This created the effect of anticipation and follow-through that was not there in the original data. Another part of their method is a calculation that slightly time shifts the LoG filter for the boundary points of an object. This time shift caused the character to reach positions earlier

or later than in the original motion data, which deformed the character's geometry, creating squash and stretch and resulted in a more exaggerated motion. The animations created by the cartoon animation filter produced satisfying results and could affect other forms of input data, such as a 2D image sequence, but it is a one parameter system that gave the animator little control over the change in data. This lack of user input may not be practical when manually animating a 3D animation.

Kwon and Lee [8] created a two-part filter that used motion capture data and converted it to rubber-like exaggerated motion. The first part of the filter created trajectory-based motion exaggeration, a distorted version of their character's motion created by stretching the trajectory and link constraints of each joint in 3D space. However, the stretching could create errors due to the change in each links length. To minimize errors, they used an algorithm called *Fast Joint Hierarchy Correction*. This algorithm measured how far the new trajectory exceeded the original link constraints length, then corrected each joint position in the hierarchy. This process is iterative and continued until it reached the chain's root joint. This decreased the total link length after each iteration to adjust the margin of error. They could also modify the error threshold to control the amount of exaggeration. The second part of their filter divided these joint links into small, equal length unit joints that are used to mimic the bending effect of rubber. This new set of joints became a sub-joint hierarchy of the original joint chain and could change the exaggerated motion into a rubber-like motion using Bézier Curve Interpolation to reposition the sub-joints. By controlling the joint structure and their links while providing a sub-set of joints to reshape

the geometry, their method was able to alter realistic motion data to create a rubber-like exaggeration.

Kwon and Lee [4] created another filter that simulates squash and stretch by optimizing spatial and temporal data from motion capture data. They specified that the filter altered realistic motion data because cartoon animation requires motion that is based on realism, but these cartoon motions cannot be achieved in a realistic environment. Their filter calculated squash and stretch poses by using a time-warping function on the position data of a character's individual joints. This allowed them to stretch the geometry of a character at the appropriate time by having a joint reach a certain position earlier or later in time. Figure 5 shows an example of motion data retargeted for a 3D character by two methods: the cartoon animation filter created by Wang et al., and Kwan and Lee's time-shift filter, and how each method effected the initial motion data. Both methods developed by Kwan and Lee read in realistic motion data and altered that data in some way to produce a cartoon-like animation. However, their results do not express how this filter would affect a manually created 3D animation or if it could be used to alter a character's motion during the animation process.

Savoye [6] created a motion capture system that filtered a rigid-skeletal data's Euler and Euclidian representations to create a more cartoon-like output. Unlike other techniques, this method added non-rigid effects to an existing captured skeletal structure to create a more cartoon-like animation. They use shearing and stretching distortion to apply these non-rigid warps to the skeletal topology. Before applying these warps, they refered to the original motion data's skeletal structure as Euclidian joint coordinates. These coordinates

are numeric references for each joint that they can alter. They use a Euler filter to alter each joint within the skeletal structure by rotating their numeric references. By using these mathematical representations for positions and rotation, the artist could apply warping features to a rigid-skeletal shape making it more cartoon-like. Figure 6 shows the change in a model's gait after applying these calculations. This technique created squash and stretch effects independently of a character's skin layer by altering the physics of a rigid skeleton. However, it focused on using motion capture data as the input and, like Kwan and Lee, it does not mention how this technique affected hand-made animations.

Figure 5: Comparison of exaggeration using Wang et al. animation filter (top) and Kwan and Lee's time-shift filter (bottom). Reprinted from *The squash-and-stretch filter for character animation* [4].

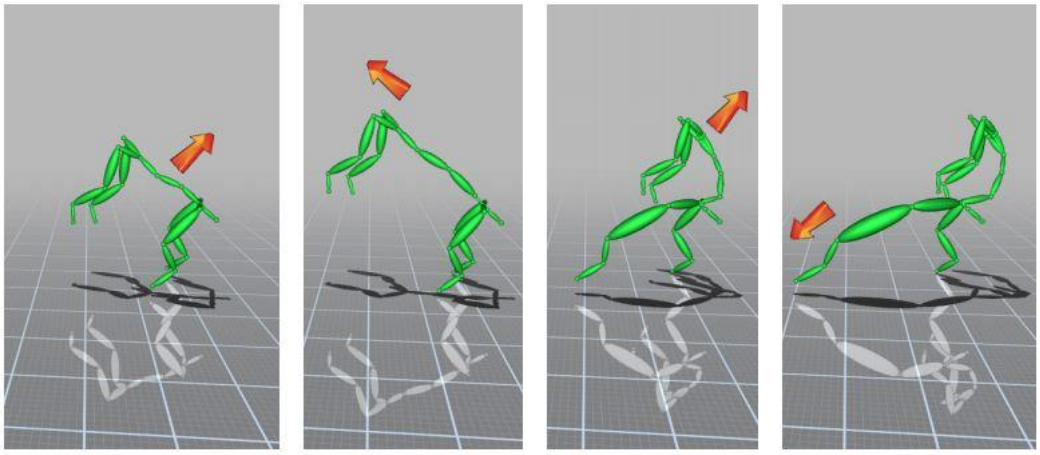

Figure 6: Result of gate becoming more cartoon-like through warping the characters skeletal structure. Reprinted from *Stretchable cartoon editing for skeletal captured animations* [6].

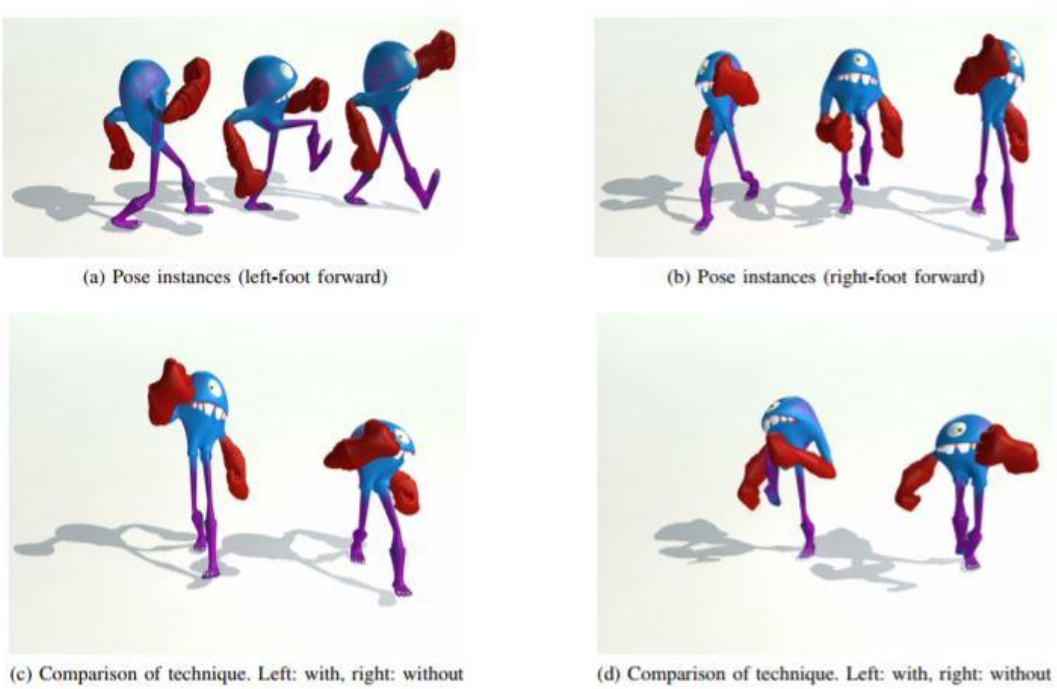

Figure 7: Results of changes to a walk cycle through poses using an example-based technique. Reprinted from *A pose space for squash and stretch deformation* [5].

Ansara [15] used an algorithm to adjust the animation curves of motion capture data to exaggerate realistic motion. This algorithm allowed the user to adjust the motion capture data's high and low points. These points reflected the animated character's joints maximum and minimum rotation. Once the user had determined these new points, the algorithm adjusted the original motion data using cubic interpolation to create a new animation. The data was adjusted using a 2D curve-based user interface to quickly edit and visualize the changes applied to the animation. It consisted of a gray curve that showed the motion data's default position, and a red curve that allowed user adjustments. Both curves are displayed along a 2D grid in which X denotes time and Y denotes the amount of change. The user manipulated these curves using arrows that translated along the grid. The further the arrows moved along the Y axis, the more intense the maximums and minimums become; moving further along the X axis slowed the animation down. Their algorithm was effective in reducing the realism of their motion data and user studies found the curve-based interface easy to use and understand. However, their cartoon-like animation is not a result of squash and stretch; the exaggeration of joint rotations resulted in more expressive poses without geometry deformations.

Another system developed by Guay et al. [10] allowed animators to sketch coordinated motion using a single stroke called the space-time curve that affected the character's animation. This technique used a matching algorithm to drive the motion of a 3D character along the space-time curve by computing a dynamic line of action. This line of action drove the character's motion along the drawn curve. The line of action did this by affecting a specific body line in the character model (spine, torso, tail etc.) and having that line move

across the space-time curve. Figure 10 shows an example of this space-time curve and line of action. Both the line of action and space-time curve could be adjusted by over-sketching or adding secondary lines. These secondary lines could also be used to affect other body-lines, such as wings on a dragon. The speed at which the space-time curve was drawn directly affected the model's form by squashing or stretching the character's skeleton (the skeleton stretched if the curve is drawn fast and squashed if the curve is drawn slow). To add further control, they added what they call space-time *cans*. Since orientation is difficult to determine in a 2D space, these *cans* direct the twisting character's orientation based on how they are placed along the curve. Through user studies they were able to determine that this technique produced animations faster than using keyframe animation techniques. It was also useful to users who have never animated before. However, they note that this technique is only useful for simple characters with few body parts due to their tool focusing on the motion of a single line.

Another technique developed by Bregler et al. [13] captured motion from traditionally animated cartoons and retargeted it onto 3D models, 2D drawings, and photographs. The input data was a video file containing the motion and a user-defined set of key-shapes chosen from the video. The video file was then turned into motion data and the key-shapes inputted by the user are connected to output key-shapes set by the user for the new character. The motion data was then mapped to the new character using the reference key-shapes, maintaining the time and motion from the original source. Figure 9 displays the results of their method using a 2D reference to pose their 3D model. By using already

established animations or drawings, the artist could produce cartoon-like movement for their own characters.

Li et al. [9] developed a similar system that used artist sketches to repose and simulate squash and stretch of their characters. This technique allowed an artist to redraw key features of an animation, such as silhouette curves over the rendered images of the original animation. The drawings were then integrated into the animation by altering the model's skeleton to match the drawn pose, and then warping the model's geometry to match the shape. Both of these techniques (Bregler et al. and Li et al.) provided a direct medium for artists to express creative opinion by taking into account their own artistic preferences. They also produced satisfying cartoon-like motion if the input material used is of good quality animation. However, if the artist wished to change the affect applied to the 3D model they would have to create a new set of 2D drawings for the system to analyze and make changes. Neither of these techniques contained the functionality to quickly adjust poses within a three-dimensional medium.

All of these techniques produced satisfying, exaggerated, and cartoon-like animations by enhancing the path of motion, deforming the skeleton of their character, or by deforming the geometry to create squash and stretch. However, some of these techniques do not discuss how their functions would affect a hand-made animation or even if it could use such data as an input. Those that could use an animator's work as input data overwrote the original animation, requiring the animator to create a new input if the new, edited versions were unsuccessful. Also, some of these methods would populate a scene with all created references, which could negatively impact the tool's performance. The Stretch-Engine's

design is focused on creating squash and stretch in realistic motion but also to be used within the animation process. This secondary goal sets a standard for the method to be lightweight and to provide straightforward controls.

When studying these methods to exaggerate animation some of them stood out. These methods included the non-realistic methods developed by Guay et al. [10], Ansara [15] and the realistic method developed by Shapiro and Lee [11]. Each of these systems contained a visual reference that allowed the animators to view exactly how their input was affecting the animation. They displayed curves that related to either the animation curve data or the paths of motion for their characters, and then allowed the animator to alter those curves to create new results. By combining a design that implements the use of interactive curves that alter animation and the flexibility of a non-realistic approach for geometry deformation, the Stretch-Engine can achieve both of its goals by controlling the changes in squash and stretch by manipulating a character's motion paths.

Figure 8: Comparison between original and edited motion data. Changes made using the Curve Interface (right), Default curve (grey) and Modified Curve (red). Reprinted from *Adding Cartoon-like Motion to Realistic Animations* [15].

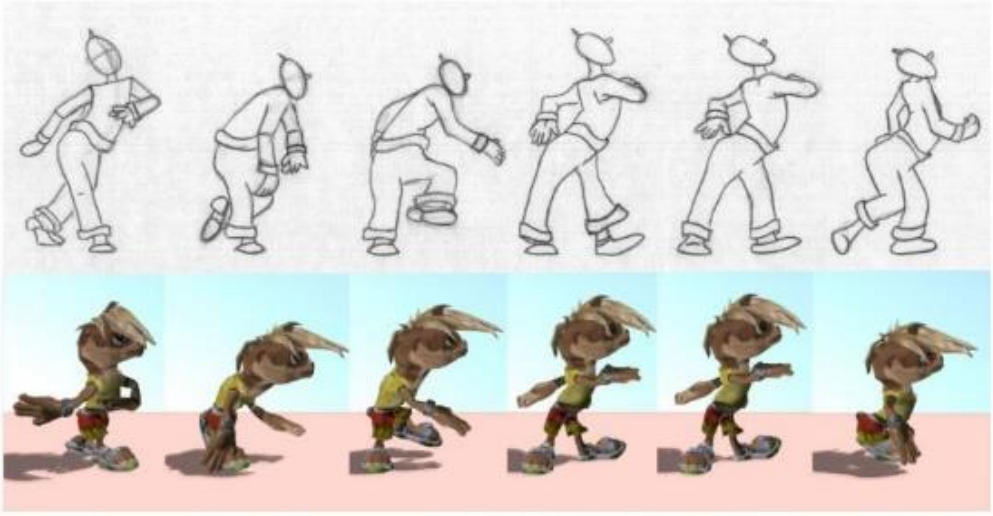

Figure 9: Example of a 2D drawing applied to a 3D character to create a stylized walk. Reprinted from *Turning to the masters: motion capturing cartoons* [13].

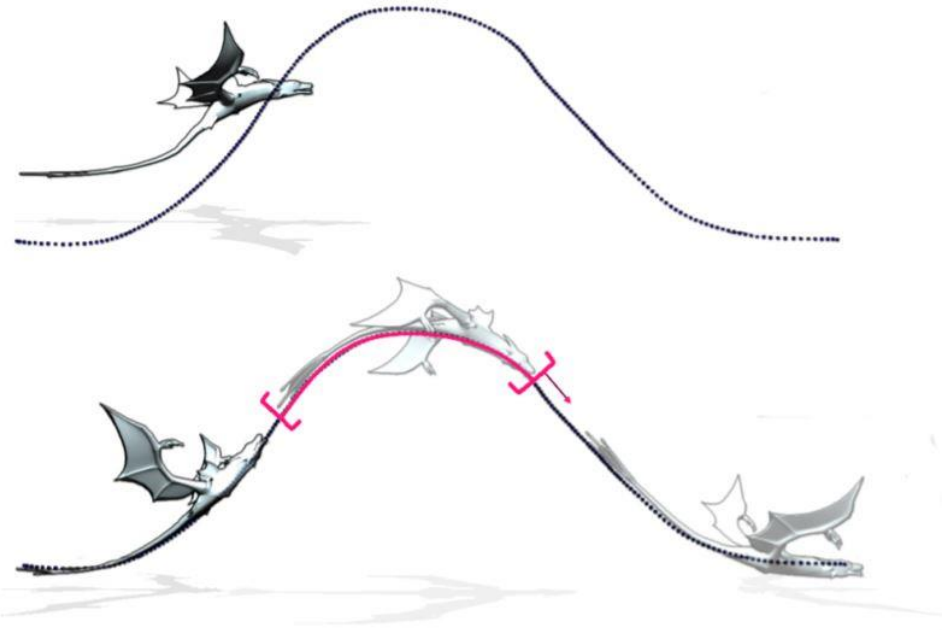

Figure 10: The effect of the space-time curve on a 3D character's motion. The warp (red) produces a line of action along the drawn line that the model follows. Squash and stretch is controlled by how fast the line is drawn, the faster the more stretched the model. Reprinted from *Space-time sketching of character animation* [10].

3. METHODOLOGY

This paper outlines the development of a prototype tool that controls squash and stretch and applies these changes to an animation. The goal is to create exaggerated character motion by giving the animator control over changes in a character's form. The tool developed to demonstrate this goal is designed to work within the framework of an existing animation software, Maya, which is to be used within a production environment. Previous tools, such as the ones mentioned in the related works section, either create realistic animations using physics-based calculations or create exaggerated animations that produced a specific cartoon-like results. The tools in the related works section were studied and, from this study, four research objects were created.

3.1 Objectives

1. To design a bipedal-humanoid rig that is capable of performing squash and stretch along each limb and contains controls for animation. This rig excludes facial animation, as the method focus is exaggerating body motion.
2. To observe and record changes in known animations to determine specifications of a prototype tool. This study focuses on the animated series "The Looney Tunes Show."
3. To develop a prototype tool in Python that creates squash and stretch in the rig while giving animators control over these changes using a motion path-based controller.
4. To provide results demonstrating exaggerated versions of animations based on realistic animations created using the prototype tool.

4. IMPLEMENTATION[2]

This project describes a method that allows animators to control changes in squash and stretch to create exaggerated motion. The Stretch-Engine is a prototype built to test this method and consists of three parts, the first part consists of a bipedal character rig that can perform squash and stretch. The second part is a set of observations determined from studying Looney Tunes animation during moments of squash and stretch. The third part is a user interface with the ability to adjust animation using 3D curves that show a limb's path of motion. The animator can use the interface and curves to control the squash and stretch of a character's limbs by scaling and manipulating the shape of these new curves.

The prototype is developed using Python and MEL commands to integrate smoothly with the Maya animation system. Python is a widely used programming language for general-purpose programming. Python has a design philosophy that emphasizes code readability and contains a syntax that allows programmers to express concepts in fewer lines of code than other languages, such as C++ and Java [24]. Python scripting can be used for many tasks in Maya, from running simple commands to developing plug-ins, and several different Maya-related libraries are available for different tasks.

The Maya Embedded Language, or MEL, is a scripting language used to simplify tasks within the Maya interface. MEL offers a method of speeding up complicated or repetitive

[2] Parts of this section are reprinted with permission from "The stretch-engine: a method for adjusting the exaggeration of bipedal characters through squash and stretch" by Zaid H Ibrahim, 2017. In Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '17), Stephen N. Spencer (Ed.). ACM, New York, NY, USA, Article 30, 2 pages. DOI: https://doi.org/10.1145/3099564.3106639

tasks by executing commands within Maya's command line [23]. This language allows tools built with it to use functions developed within the Maya interface. Most tasks within Maya's graphical user interface (GUI - a visual way of interacting with a computer using items such as windows, icons, and menus) can be performed using MEL commands as well as some that are not available within the GUI.

The flexibility of Python and the variety of commands provided by MEL make them the most appropriate languages for the Stretch-Engine code. The Stretch-Engine code is made up of two major scripts, the Stretch-Engine Rig and the Exaggeration Interface.

4.1 The Stretch-Engine Rig

When first developing the Stretch-Engine, research focused on creating the Stretch-Engine Rig. For the rig to perform optimally it would need to be simple to use and able to perform deformations required for squash and stretch. The structure for the rig would also need to reflect characters from "Looney Tunes" animation as they were the subjects used to create a range for stretching. Due to the structure requirements, the rig is a bipedal rig and reflects the general form of Looney Tunes characters like Bugs Bunny and Daffy Duck. The design of this bipedal rig can be broken down into three sections, the joint structure, the IK systems required for limb motion, and the controls for each limb.

The first step in creating the rig's joint structure is generating locators. Locators are objects within the Maya interface that save transformation data such as position, rotation, and scale. These locators create the position and rotation data that the joints use when being

created. The use of locators in generating the rig was to allow the user to adjust the joint positions and fit them to any model they wish to use for animation.

The joint structure contains minimal sets of joints for simplicity and to optimize efficiency when running the rig script. The original position data for the locators is set to match the Stretch-Engine Dummy, a model that is used in this thesis to animate the motion studies and view the Stretch-Engine results. The model reflects a bipedal character structure and is designed with the intention to perform squash and stretch. Figure 11 shows the locator, joint and geometry structure of the Stretch-Engine. Once the locators are in place the user can run the rig script to build the joints, IK systems, and controls.

The rig's IK system is made with simple, rotation and spline IK solvers. These solvers are defined in the Maya interface and are used for different forms of transformation. The rig's arms and legs use a rotation IK solver, allowing a pivot to be placed that creates motion such as turning the elbows and knees. The neck uses a simple IK solver to allow the head to translate while moving the neck in the respective direction. The spine uses a spline IK solver, allowing multiple controls to be placed along it to create the diverse motions of a person's torso such as arching back and bending over.

Each limb section (arms, legs, neck, and spine) needs to perform deformations for the desired changes in squash and stretch. To create changes in form without any internal calculation issues, each limb section contains a separate joint structure that is bound to the geometry. While the IK joints for each limb determines motion, these *Binding* joints scale and deform the geometry. This allows the binding joints to squash and stretch the geometry

while keeping the calculations for the IK systems unaffected, allowing the original joints to perform proper limb motion.

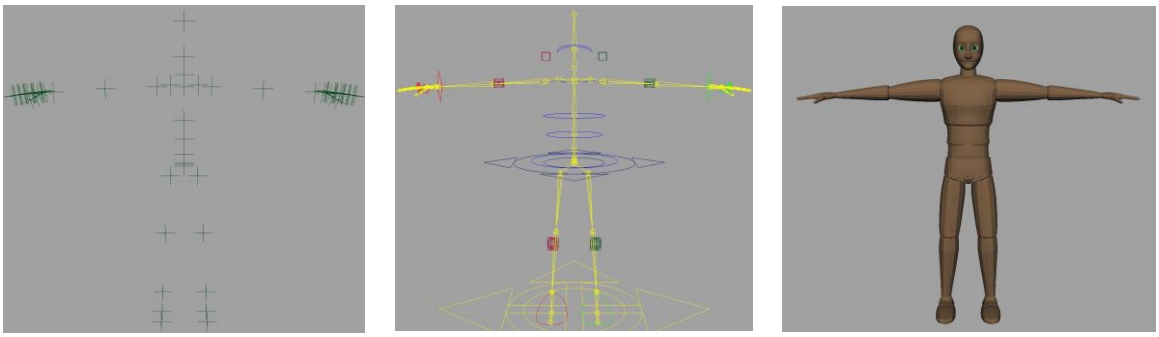

Figure 11: The Locator Structure, Joint Structure and Stretch-Engine Model.

Each limb structure contains a distance ruler, an object in Maya that measures the distance between two points, to create the changes in squash and stretch. To properly calculate the length of each limb the ruler's start and end points need to cover the limb's entire joint structure. Figure 12 displays the distance ruler along the left arm's joints. The area covered for the arms is from the shoulder to the wrist, the legs are from the hip to the ankle, the spine is from the spine's base to the spine's tip, and the neck is from the base of the neck to the base of the skull. To maintain an accurate distance of the limbs total length, the recommended starting position for each limb is where the limb is completely straight.

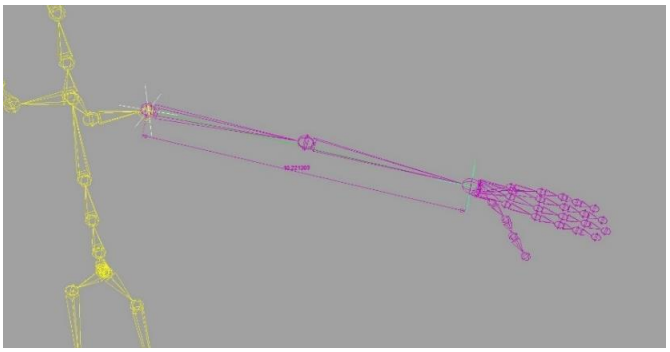

Figure 12: The IK and Distance Ruler along the Left Arm joint structure.

This paper uses the arm structure as the primary example when discussing the distance ruler and its effect. The arms joints are scaled to create the physical change in the arm. When the distance between the shoulder and the wrist extends, or contracts, the arm stretches and squashes respectively. The fraction of change between the new position and old position is a numeric representation of the change in squash and stretch. This fraction is then applied to the rig joints, scaling them and emphasizing the change in form along the joint structure's longitudinal axis. This means if the arm is moved past its default straightened length it changes shape. If the arm's size is greater than the default length the arm stretches along the arm's length and if this size is less than the default length the arm squashes. Figure 13 shows how this change in position affects the left arms geometry.

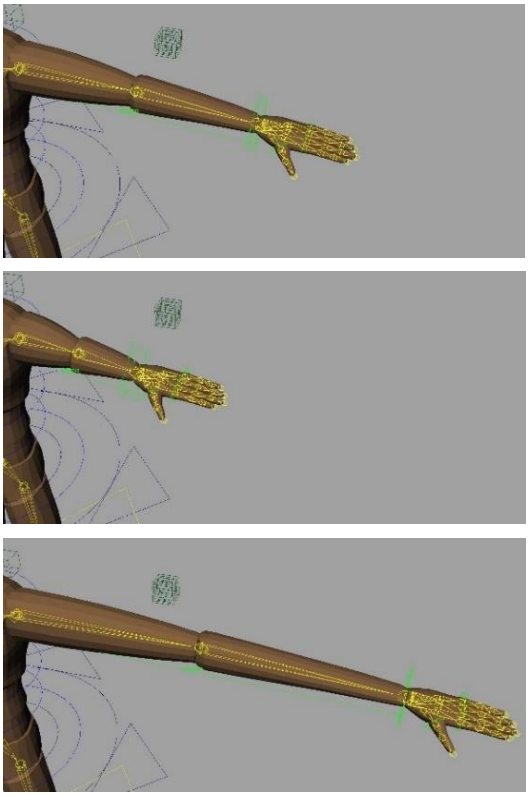

Figure 13: The squash and stretch effect in the arm as the controller moves passed the default length. Default length (top), Squash length (middle), Stretched length (bottom).

To provide more flexibility, the fraction of change is also used to calculate an approximate amount for volume preservation. This amount inversely affects the arm's width and height compared to the change in length. Figure 14 shows an example of the volume preservation along the left arm. This effect is similar to stretching a rubber band; the further the arm extends, the thinner it becomes and the more the arm is compressed the thicker it becomes. The rig is meant to give the animator as much control over the resulting animation as possible, and although volume preservation is not prominent in Looney Tunes animation, this feature is provided as an extra form of control for animators to use in their animation.

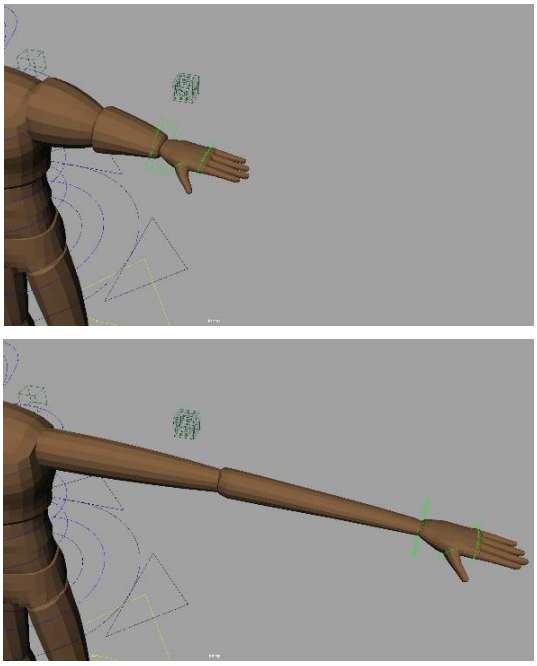

Figure 14: The volume preserving feature in the character's arm. The arm becomes thicker as the limb squashes (top) and thinner as the limb stretches (bottom).

The controls are the final elements added to the rig. These controls, shapes made using Maya curves and attached to sections of the rig, allow the animator to manipulate the rig

structure and create their desired animation. The controls also contain attributes for more defined animation poses, such as curling the hand into a fist or rolling the ball of the foot. Each limb controller also contains attributes for switching the squash and stretch and volume preserving features on and off. The controls are designed to be simple so as to not distract the animator or block the model's geometry. This way the animator may focus on creating their desired animation while being able to view the changes they make. The *Stretch-Engine Rig* is integrated into the Stretch-Engine's user interface for quick access and to assist with the workflow for creating squash and stretch.

4.2 Cartoon Observations

"Looney Tunes" animations from the years 1940 to 1960 were studied to find a *range of scale* that determines the amount of squash and stretch available to users. The limbs of Looney Tunes characters, such as Bugs Bunny and Daffy Duck, were measured during relaxed poses and stretched poses to find this range. The animated character's limb length during its relaxed pose is set as their default limb length, and any alteration during exaggerated actions to their limb's size was considered for their stretch length.

Each episode was studied frame by frame to measure the full extent of changes in the subject's limbs. Due to the multiple sources for these videos, standard measurements of length did not carry over from one video to another. For this reason, the unit of measurement used in these observations is based on the character's arm length in the current episode (i.e. one unit equals the length of the characters arm in the default position). Any change measured is in relation to this unit of scale to create accurate limbs deformation

values. Table 1 shows the measurements found for each character during exaggerated moments and the episodes of Looney Tunes animations used in the study.

Episode Name; Director(s); Date Released	**Character**	**Default Limb Size (DLS)**	**Action**	**Stretch Limb Size (SLS)**	**Change = SLS/DLS**
Duck Amuck; Charles M. Jones; *Feb 28, 1953* [16]	Daffy Duck	**Arm:** 1 unit **Torso:** 1 unit **Legs:** 0.67 unit	Angrily Jumping Around	**Arm:** 2 unit **Torso:** 2 unit **Leg:** 1.34 unit	**Arm:** 2 **Torso:** 2 **Leg:** 2
Fast and Furry-ous; Charles M. Jones; *Sep 17, 1949* [17]	Wile E Cayote	**Arm:** 1 unit **Torso:** 1 unit **Legs:** 1 unit	Painting Wall	**Arm:** 2.25 unit **Torso:** 1.75 unit **Legs:** 1.5 unit	**Arm:** 2.25 **Torso:** 1.75 **Leg:** 1.5
Tortoise Wins by A Hare; Robert Clampett; *Feb 20, 1943* [18]	Bugs Bunny	**Arms:** 1 unit **Torso:** 1.4 unit **Legs:** 1 unit	Complaining about Tortoise	**Arms:** 2.4 unit **Torso:** 1.82 unit **Legs:** 1.6 unit	**Arm:** 2.4 **Torso:** 1.3 **Leg:** 1.6
The Great Piggy Bank Robbery; Robert Clampett; *July 20, 1946* [19]	Daffy Duck	**Arms:** 1 unit **Torso:** 1 unit **Legs:** 0.5 unit	Scared by Neon Noodle	**Arms:** 2 unit **Torso:** 1.5 unit **Legs:** 2 unit	**Arm:** 2 **Torso:** 1.5 **Leg:** 4
Stop! Look! and Hasten!; Charles M. Jones; *April 30, 1955* [20]	Wile E Cayote	**Arm:** 1 unit **Torso:** 1.25 unit **Legs:** 1 unit	Pouncing to catch a fly	**Arms:** 2 unit **Torso:** 1.4 unit **Legs:** 2 unit	**Arm:** 2 **Torso:** 1.12 **Legs:** 2
What's Opera Doc?; Charles M. Jones; *July 6, 1957* [21]	Bugs Bunny	**Arms:** 1 unit **Torso:** 1 unit **Legs:** 1 unit	Dancing with Elmer	**Arms:** 1.4 unit **Torso:** 1.4 unit **Legs:** 1.4 unit	**Arm:** 1.4 **Torso:** 1.4 **Leg:** 1.4

Table 1: Results of studying Looney Tunes episodes and how their characters were stretched to exaggerate their actions. A "Unit" is the size of the character's arm, from shoulder to wrist, during rest pose.

After averaging Table 1's Change value for each limb, we see the *average change per unit* is 2.008 for the arms, 1.512 for the torso, and 2.083 for the legs. Based on these results and from studying other episodes of Looney Tunes animation, four common traits were found. These traits have become key observations in the development of the Stretch-Engine's range of scale:

1. When a character's limbs are stretched for the exaggeration of an action, they become approximately twice their default size.
2. Torso stretching seems to compliment the change in limb size rather than make up the entirety of the exaggeration. Resulting in a torso stretch less than or equal to the stretch of the limbs.
3. Characters do not squash unless they are under the influence of another object, such as an anvil falling on top of a character or being hit with a heavy object.
4. In extreme cases such as being pulled by a rocket, the character's limbs stretch to approximately five times their default limb size.

These observations determine that an appropriate range of scale for squash and stretch falls between zero, when a limb has become completely squashed, and five, in case of an extreme change in stretching. When set to a range of two for example, the limb reaches twice the default length at its most extreme position. The observations also show that different limbs scale differently based on the desired intensity of action, such as the case for the torso.

The range of scale found in this study is used to create the exaggerated path of motion. The exaggerated path is used to create the desired change in a limbs motion path and is the key controller in squash and stretch provided by the Stretch-Engine.

4.3 Functionality of the Stretch-Engine

4.3.1 Curve Development

When first developing the curve creation function for the Stretch-Engine, the built-in function *Create Editable Motion Trail* within the Maya interface seemed to be the best solution in providing animators with a control and visual reference for motion. This function allows an animator to create a motion trail for the selected object and alter the objects position along this line. Each keyframe for the object is represented by spheres, or *timing beads*, along the curve. As the animator moves these beads in space, the curve and object update to match this new position.

The animator can specify how much of the animation the trail affects by setting a range of frames or the entire timeline. The trail also contained an increment value that changes how the trail is drawn and how the position data is sampled. For example, if the increment is set to 1 every frame is sampled and given a point along the curve while keyframes in this timeline are given a bead, but if set to 5 every fifth frame would be sampled and placed along the curve but even if a keyframe does not fall along this line that keyframe is still given a sphere.

Maya's editable motion trail also contains a few extra features such as pre and post frames, always draw/draw when selected, trail thickness, key size and show frame

numbers. These features affect how the curve is drawn and how the animator views the editable motion trail. Setting a value for pre and post frames has the curve drawn that many frames before or after the current frame. This gives the curve an effect of being drawn and erased as the animation plays. The attributes “always draw” and “draw when selected” are the curves “Pinning” type.

The term “Pinning” does not fully explain what function this feature has on the editable motion trail. Maya defines this term as how the motion trail is drawn in the scene. Setting it to “always drawn” has the curve displayed even when its respective object is not selected, and “drawn when selected” displays the motion trail only when the respective object is selected. The last three attributes are cosmetic changes that affect the editable motion trail. The “Trail Thickness” attribute adjusts the drawn curve’s thickness while the “Key Size” attribute affects the size of the keyframe representations along the curve. The “Show Frame Number” attribute adds geometry to the trail to show keyframe numbers above the respective keyframe along the trail.

These features give the animator a variety of ways to control Maya’s motion trail to match their specifications. The motion trail also smoothly updates the selected object’s position while providing a visual representation of the change in space. It seemed the most viable option for the Stretch-Engine, however after some tests it was found that the trails shape cannot be affected in any way other than moving the beads, providing two major problems.

The first problem was that moving the beads directly affected the objects position. This made it difficult to provide a preview of the change in animation as the curve was

immediately affecting the position data. The second was that the curve could not be scaled, a feature that is an important part of the Stretch-Engine's design concept. The exact change in the limbs motion path to create the required squash or stretch can be displayed by creating a secondary curve, a scaled version of the original curve. Based on these observations, the motion trail used in the Stretch-Engine was built from scratch and designed to satisfy a set of goals.

These goals were created based on studying Maya's editable motion trail, and incorporate some features found in the trail and ones that were missing:

1. The motion trail created for the Stretch-Engine would need to accurately reflect the selected limb's path of motion.
2. The trail would need to be allowed to change shape without immediately repositioning the controller.
3. The trail scales and changes the selected limb's size based on the *range of scale*.
4. Like Maya's motion trail, the Stretch-Engine's trail can be edited manually to provide another degree of control over the resulting animation.

Based on these goals (providing an accurate visual reference, the ability to preview changes in animation, to change the selected limb length by the desired scale, and to give the animator manual control over the curve) the Stretch-Engine generates a three-dimensional motion path with a structure that is based on the selected limb's keyframed controller positions, its path of motion. This curve is called the Exaggerated Motion Path

and it gives an animator the ability to adjust exaggeration through squash and stretch by changing a limbs path of motion.

4.3.2 The Exaggerated Path of Motion

To give the animator a point of reference, another path is created called the original path of motion. This path has no effect on the animation other than visualizing the selected controllers keyed animation before changes are made. The exaggerated path of motion is created similarly to this original path, but the key difference is its ability to alter and scale the exaggerated path using controllers that are provided to the user. The first step in creating this curve is to find the keyframe data for the desired timeframe. This data is queried using the following function:

```
keyframes = sorted(list(set(cmds.keyframe(obj, q=True, time=(startFrame,endFrame), timeChange=True))))
```

This code queries the selected range of time using the *startFrame* and *endFrame* inputs and finds all keyframes within the range specified for the selected object, obj. This returns a set of frames where keyframes exist but the set is unordered. The set is then placed within a list and the list is sorted to return an ordered list of frames from earliest to latest. The order of these keyframes specifies the curve shape and in doing so sets the created curve's origin point as the *start* keyframe's data.

An option is available to reverse the curve, this sets the order of frames from *end* to *start* in the case that the animator would like the curves origin point to begin from the final frame. This allows scaling to occur in the opposite direction of motion, which is useful for

actions where the limbs are being pulled away, such as the case when a Looney Tunes character is being dragged offscreen by a rocket or some form of outside force.

To obtain the selected controller's transformation values the code iterates through the sorted list for its entire length. It does this by moving through the frame of time specified by the *start* and *end* frames to each of the keyframes within the list, and the selected controller's position is queried at the respective moments in the timeline. These position values make up the control points, these are points along the curve that make up the path of motion's shape. The controller is then set to follow along this curve by using a locator.

The controller is attached to the motion path by a locator that is keyed to each curve's control point. This sets the locator to move in space along the curve as the timeline is adjusted. The controller follows this locator in space so that it too can move along the curve. This creates an animation based on the curve's shape. Both the exaggerated path and the original path of motion consists of one of these locators and these locators both contain influences on the selected controller. The influence on each curve can be adjusted by the system, allowing the animator to view the new animation along the exaggerated curve while also being able to easily return to the original path of motion.

The exaggerated path of motion is an interpolated version of the original path of motion and a scaled version that takes into consideration the range of scale set by the animator. This interpolation can be affected by *easing*, an effect that makes animations feel more natural through easing-in, slowly starting and accelerating, or easing out, quickly starting and decelerating. Easing provides a way for the animator to control the change between the original length and the new length from the exaggerated version. For more control over

this path of motion, the animator can set the motion path scales along the X, Y and Z axes of the exaggerated path. This is important in actions that only need to be stretched in one or two degrees of motion rather than apply a change in all degrees.

The exaggerated path of motion also contains controllers or control spheres along each point within the curve. These control spheres allow the animator to change the curve's shape and give the animator the ability to control the change in squash and stretch by manually pulling the curve control spheres in the direction they desire. This can be useful for making minor adjustments to the new path of motion or to reshape the curve if the old path of motion was undesirable. When the animator has created the motion path's new shape they can then re-connect the controller to follow the new shape and can view the resulting animation.

The exaggerated curve's effect is a transition between the selected limb's original length to the limb's scaled length that takes place over the set time line, where at the starting frame the limb reflects the original path of motion and the end frame the limb reflects the exaggerated path of motion.

4.3.3 Building Exaggeration

The exaggerated path of motion works by having the final point create the desired change in size for the selected limb. This is achieved by having the distance between the controller and the base joint equal to the new length set by the scale. This position is somewhere between two points, the position of the limbs controller during the action and a new position of that controller when scaling the entire motion trail by the set scale. The

reason the point is between these two positions is because the position found when scaling the trail creates too long or too short of a change in size for the selected limb.

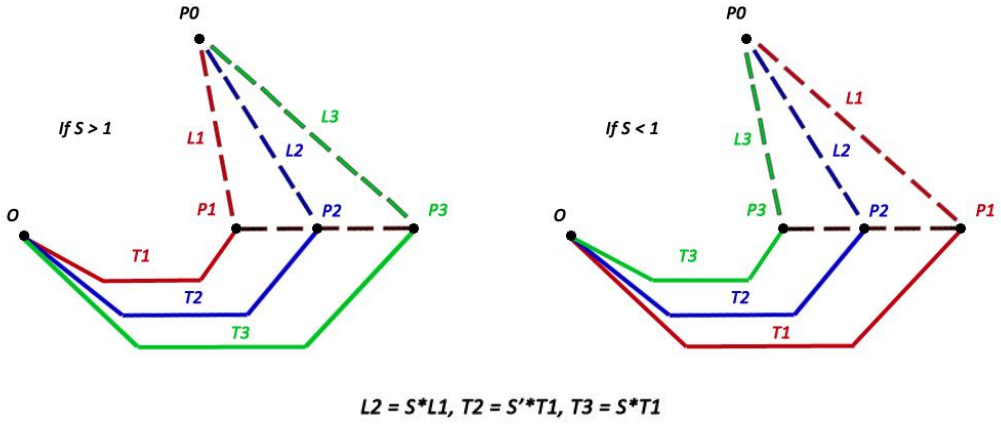

Figure 15: The two cases for finding the position of P2. The left image is how a triangle is determined for a scale greater than 1 and the right is for a scale less than 1.

To calculate the appropriate position the Stretch-Engine views the information it has as a triangle. This triangle is made up of three points; the base joint P0, the final position of the limbs controller along the original motion trail, P1, and the respective position along the fully scaled motion trail, P3. The length, L3, of the limb when at P3 is either larger or smaller than the required limb size, the scale S times the original length L1, if this scale is greater than or less than 1. There exists a motion trail scaled by an unknown value, S', that has a final point P2 that creates the change in length L2 that equals S*L. This point is somewhere between the points P1 and P3 and along the line made by these points. Diagram 1 shows the two separate cases based on the set scale S and the triangles created from the motion trails.

To calculate P2 the Stretch-Engine breaks these triangles down to the smaller triangle with P2 as one of the corners and L2 as one of the sides. It then uses the Law of Sine to find the coordinates of P2, however before we can use the Law of Sine all angles within the smaller triangle need to be identified. The angle B is the largest angle within either triangle and is the angle between vectors V0 and V1. Diagram 2 shows the breakdown of each triangle and how V0 and V1 change based on the scale.

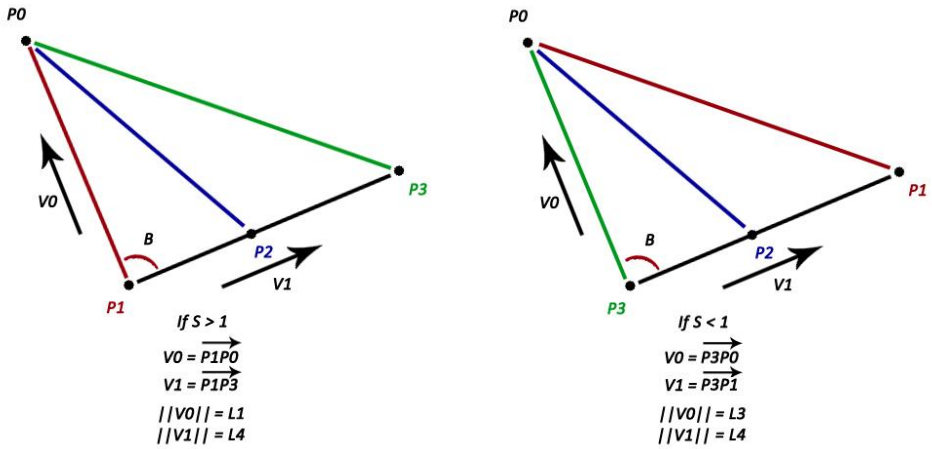

Figure 16: The resulting triangles created from both scale cases. It displays the vectors V0 and V1 as well as the magnitude for those vectors.

$$V0.V1 = V0x * V1x + V0y * V1y + V0z * V1z$$

$$V0.V1 = ||V0|| * ||V1|| * \cos(B)$$

$$B = \arccos\left(\frac{V0x * V1x + V0y * V1y + V0z * V1z}{||V0|| * ||V1||}\right)$$

Once V0 and V1 are calculated, angle B is calculated using the dot product. After testing the code, it was found that calculating angle B using the smaller triangles P0-P2-P1 when S > 1 and P0-P2-P3 when S < 1 provided the system with the correct position of P2. Through my research the best assessment of why this solution works is because in either

of these cases we can assure that L2 is between P1 and P3. If the triangle P0-P2-P1 was used when S < 1, the angle at P0 would be calculated as the largest angle and P2 would fall past P3, creating a much smaller length than L3. There may be a case were this solution fails however we have yet to find such a case, and for that reason this solution is used to calculate angle B.

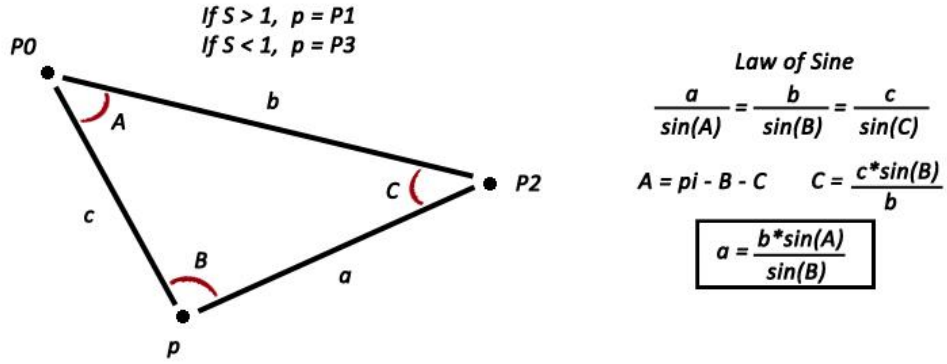

Figure 17: The angles found within either triangle used to calculate P2. Includes a breakdown of the Law of Sine calculation for finding the side a between p and P2.

Diagram 3 shows how the Stretch-Engine uses the Law of Sine and the angle B to find the length between ether P1 and P2 or P3 and P2 based on scale S. This length is then used to find P2 by multiplying it by the normalized vector V1 to find the vector V2 of P1 or P3 with P2. The coordinates of P2 can then be found by adding P1 or P3 to V2.

$$N1 = V1 / ||V1||$$
$$V2 = N1 * a$$
$$P2 = p + V2$$

$$x1 = P1x - Ox,\ y1 = P1y - Oy,\ z1 = P1z - Oz$$
$$x2 = P2x - Ox,\ y2 = P2y - Oy,\ z2 = P2z - Oz$$

$$OP1 = \sqrt{(x1*x1)^2 + (y1*y1)^2 + (z1*z1)^2}$$
$$OP2 = \sqrt{(x2*x2)^2 + (y2*y2)^2 + (z2*z2)^2}$$

$$S' = OP2 / OP1$$

The scale S is equal to the difference between length OP3 and OP1, O is the shard origin point of each motion trail. The scale S' that is needed to create the motion trail T2 and ends with point P2 is then equal to the difference between length OP2 and OP1. Once S' is found the Stretch-Engine creates a new version of the original motion trail scaled by the value. This new version is the exaggerated motion trail before interpolation with the original motion trail and the effects of ease-in and ease-out set by the animator, or the fully exaggerated version.

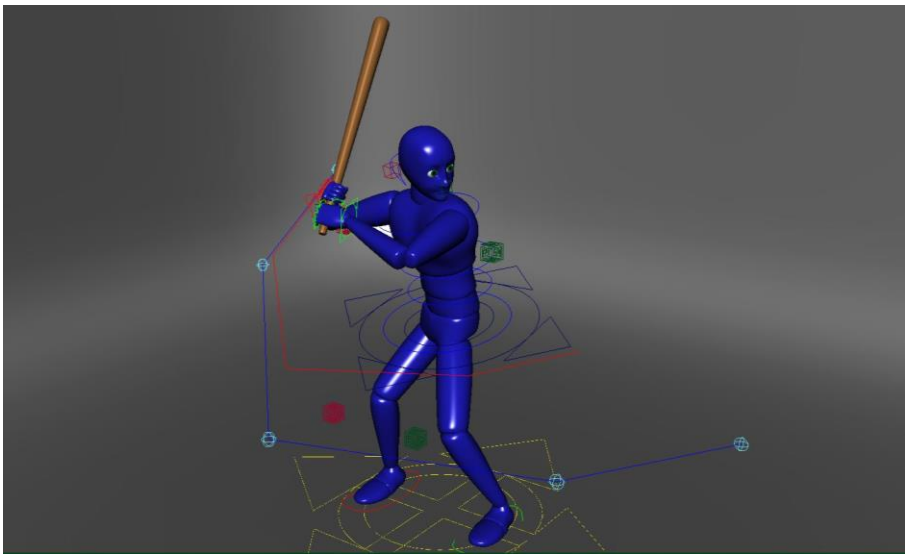

Figure 18: The Exaggerated Motion Path (blue curve) of the Right Arm. The controller path is scaled by 2 in all dimensions (X, Y, Z) with control spheres along each keyframe.

4.3.4 Interpolation and Easing

To create the gradual change from the original curve to the exaggerated curve the Stretch-Engine creates a linear scale. This scale is calculated using the points that make up the original motion trail by using the total length of the curve and the length of each point from the origin point along the curve. Diagram 4 shows an example of this concept and

how the lengths of each point are collected. By dividing each of these lengths by the total length a range of ratios from 0 to 1 is created, this becomes the linear scale.

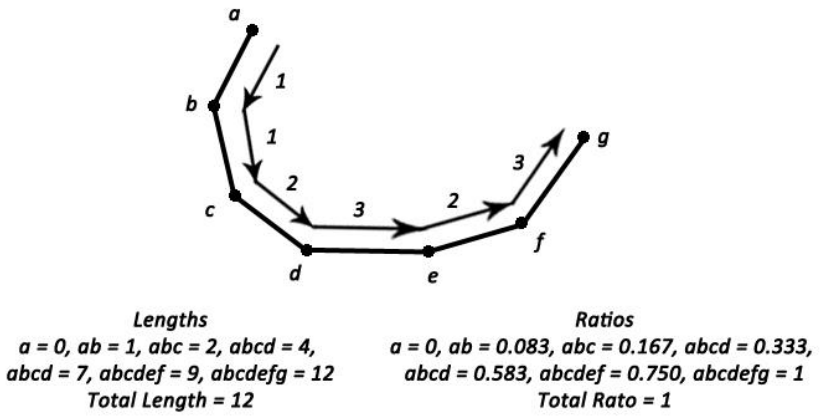

Figure 19: An example trail broken down to show the length between each point and the origin point. Along with how these lengths are calculated into a set of ratios from 0 to 1.

This ratio decides how much influence the original curve and the fully exaggerated version have on each of these points. Applying ratios to their respective points along the original version and fully exaggerated version, then adding these results together results in the coordinates of the points that make up the final version of the exaggerated curve. Points that have ratios closer to 0 lean closer to the original positions, while points that have ratios closer to 1 lean more towards the fully exaggerated positions

$$eX = oX * (1 - r) + sX * r$$
$$eY = oY * (1 - r) + sY * r$$
$$eZ = oZ * (1 - r) + sZ * r$$

Before this addition takes place, the linear scale is run through a remap tool; the remap tool is a Maya function that uses an input range and reproduces results based on a different

output range. It also contains a curve that represents how values within this range are remapped. The curve is set to a spline curve to represent the easing effect and create a more natural transition. For the purpose of the Stretch-Engine the input and output ranges are the same, 0-1. The curve is the feature of this tool that is manipulated to reassign ratio values to each point.

Through the remap, the user can apply the easing effect to the final version of the exaggerated motion trail. By selecting ease-in or ease-out the Stretch-Engine places new points along the remap curve to change its shape resulting in a new range of ratios that represents the ease effect set by the user. Diagram 5 shows the difference between a linear scale curve, the spline curve and a curve with an ease-in and ease-out point.

These new ratio values are applied when calculating the final version of the exaggerated curve. The result is a scaled version of the original motion trail that gradually changes until the final point along the trail equals to the position of the limbs controller that creates a change in length equal to the set scale.

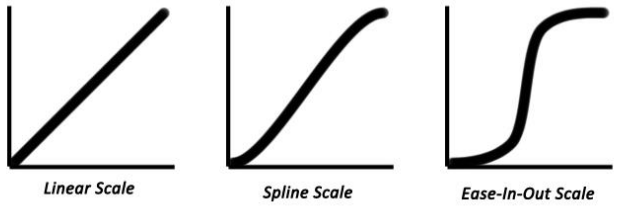

Figure 20: The differences in shape of the remap curve when using a linear curve, spline curve and an eased spline curve.

The exaggerated path of motion and the amount of easing applied to the path can be created using the Stretch-Engine interface. This interface also contains additional utilities to allow the animator to edit and control their animation.

4.3.5 Exaggeration User Interface

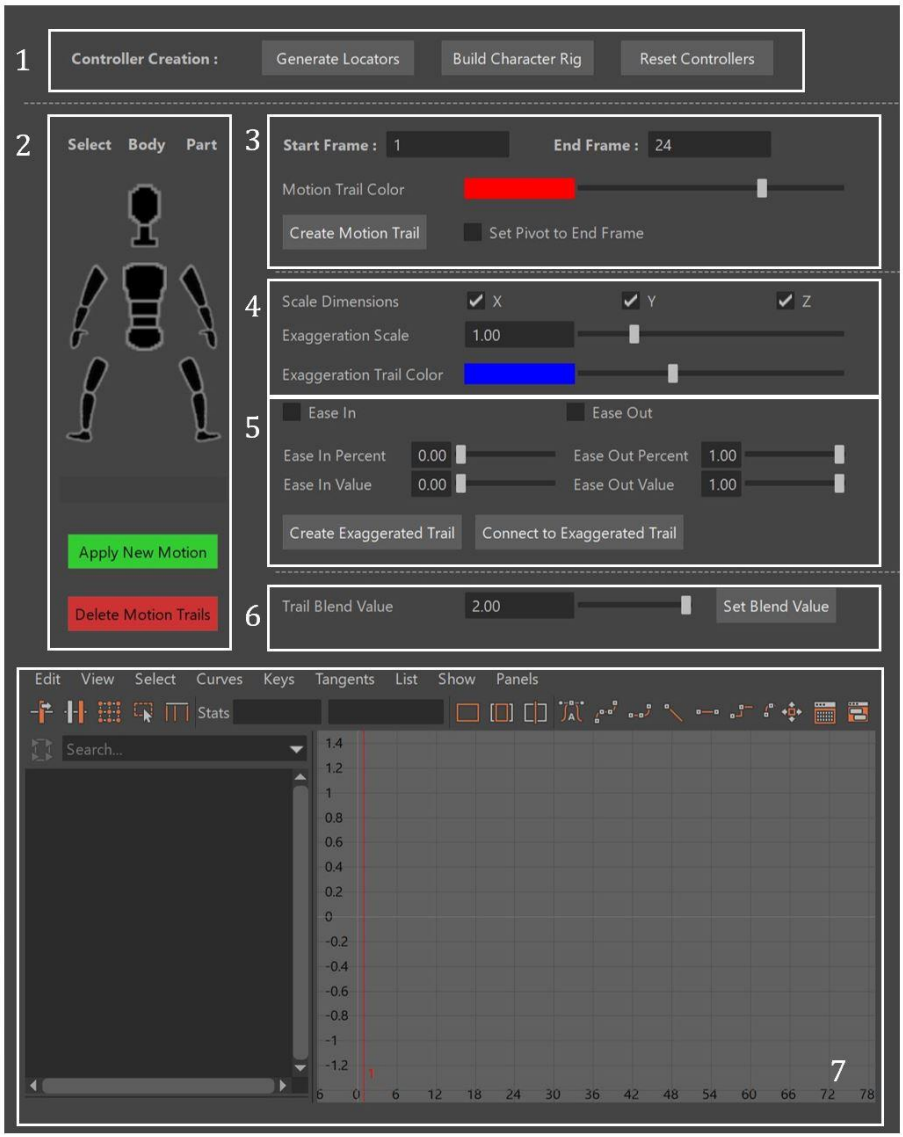

Figure 21: The Exaggeration Interface provided by the Stretch-Engine. Outlined are the 7 parts of the interface provide the animator with control over the motion trails.

The Exaggeration Interface is the main user interface for controlling the commands provided by the Stretch-Engine. It contains all the necessary controls to create the motion trails discussed in previous section that allow the animator to control changes in squash and stretch. The interface can be broken down into seven sections that are individually outlined in Figure. The next few paragraphs discuss each of these sections individually.

Section 1 consists of three buttons related to the Stretch-Engine rig. The first button generates the locators that represent the position and rotation coordinates of the rigs joint structure. The second button generates the joint structure and builds the respective IKs and controls that make up the rig. The final button provides the animator with a quick and efficient way to reset all the rigs controls to their default position.

Section 2 consists of a limb diagram for selecting the limbs respective controller within the Stretch-Engine rig. While providing an easy way to select controllers it also notifies the Stretch-Engine of the limb controller that the curves are applied to. The functions for the limb diagram are the *selectBody()* and *deselectBody()* functions. These functions select or deselect an image while identifying the limb controller that reflects the selected image. This section also contains two buttons, the first is used to finalizing the resulting animation based on the generated motion trails and the second deletes these trails, returning the animation to its unedited state. The *finalizeNewMotion()* function is run when pressing the "Apply New Motion" button. This function iterates through the timeframe selected and re-key the position of the selected controller based on the motion path it is following. The *deleteAnimCurve()* function is run when pressing the "Delete Motion Trails" button. This function deletes the curves generated within the Maya interface along with their locators

removing their influence of the selected controller and resets that controller to its original positions.

Section 3 is where the animator selects the frame range they wish to adjust. It contains a button that runs the script for creating the original motion path for this selected time range and for some extra customization there is a slider for setting the curve color. The start frame and end frame inputs are saved to global variables that are used in each function when generating curves. For the original motion path, when pressing the "Create Motion Trail" button the *createAnimCurve()* function is run. *createAnimCurve()* is a custom function developed for the Stretch-Engine. The function iterates through the timeframe and uses the sorting function highlighted in the *Exaggerated Path of Motion* section to find all times that contain a keyframe for the selected controller. For each of these keyframes it saves the position data of this controller and constructs a curve with control points based on these position values. The curve generated follows the original path of motion created by the animator and is the color of the color slider.

Section 4 contains the controls for generating the exaggerated path of motion. Here the user can select the axes they desire for the scale and the curve using check-box inputs. The user can also set the scale to be applied to the curve using a slider. This slider reflects the range of scale found in the cartoon observations and can be scaled by a value of 0 to 5, or 0% to 500% of the limb's length. Setting the curve axis and scale value are required settings to generate the exaggerated curve. The *createExagCurve()* function is run when pressing the "Create Exaggerated Trail" button. Similar to the *createAnimCurve()* this function uses the timeframe set by the animator but instead of iterate through the controllers keyframes

it duplicates the original path of motion and identifies the control vertices, CVs, within the curve. CVs are the points in space that make up the shape of the three-dimension curve. In the case of the Stretch-Engine, CVs reflect the position data of the selected controller. By moving these CVs, the animator can manipulate the shape of the curve. To give the animator this ability the function also builds controllers that are able to move the CVs of the editable path.

Section 5 contains the optional controls for exaggerated path of motion. This is where the animator can apply the effects of easing to the shape of the generated curve. Normally easing is represented by a visual representation such as the curve diagram previously used, however there were complications when trying to create such a controller. In this section numeric values represent the position along the spline curve within the remap. The curve is a 2-dimensional curve and for the explanation of the position values of the ease-in and ease-out points are expressed in terms of the X and Y dimensions. Figure 17 shows a version of section 7 with both ease-in and ease-out turned on.

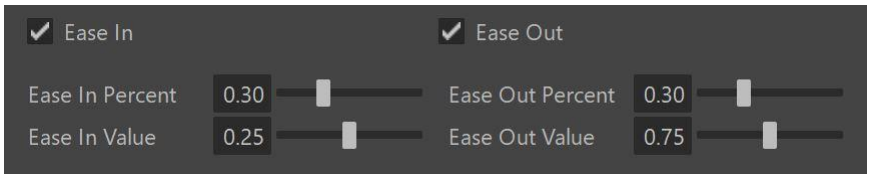

Figure 22: Section 7 of the interface with both ease-in and ease-out selected. The sliders determine the position of the point along the remap curve.

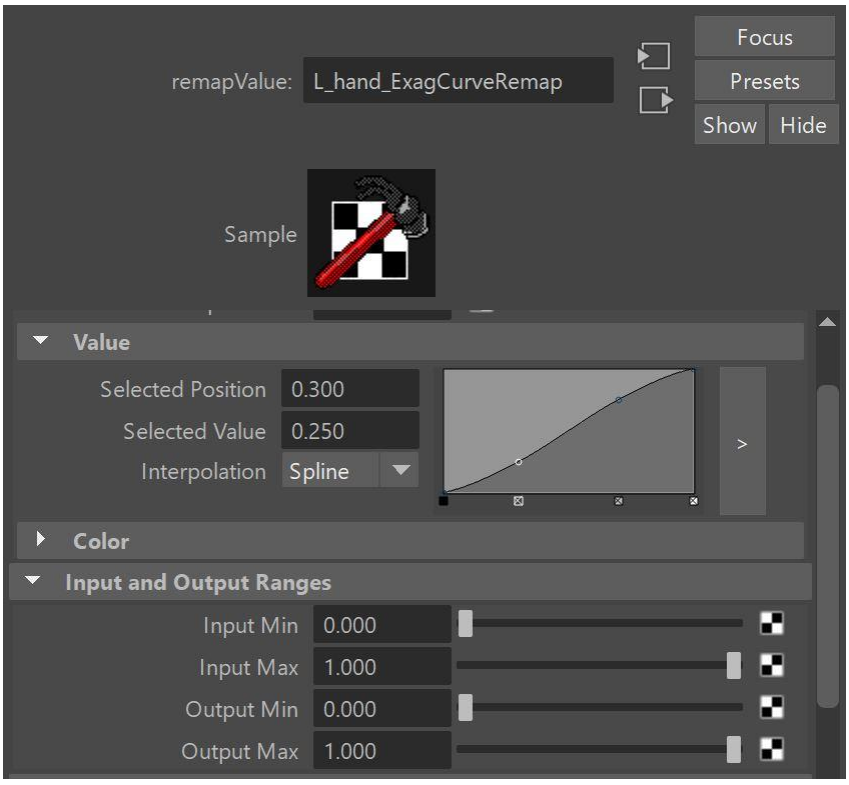

Figure 23: The Remap Tool within the Maya interface displaying the remap curve. The values to the left are the ones set by the animator for the ease-in point.

When checking either the ease-in or ease-out boxes a point is created along the remap curve. The *percent of ease* is the selected points position along the X-axis while the *value of ease* is the points position along the Y-axis. Percent states how long the selected ease is present along the curve. The larger the percent the further along the X-axis the point is. For ease-in the higher the percent the further to the right the point is while ease-out is further left. Value states how much ease affects the curve. The larger the value the higher the selected ease is along the curve. Ease-in starts lower on the curve as it slows down the change from the original trail to the exaggerated trail, while ease-out begins higher on the curve. To match their effects on the curve, ease-in has a minimum value of 0 and a maximum value of 0.5 while ease-out has a minimum value of 0.5 and a maximum value

of 1.0. Figure 19 shows the effect on the remap curve of a change in percent and a change in value for the ease-in point. When the exaggerated curve is created it runs the *setEase()* function that reads in the ease-in and ease-out fields then restructures the remap curve based on the inputted values.

Figure 24: The effect of changing the ease-in points position along the remap curve. The first shows a change in percent while the second shows a change in value.

When the animator manipulates the shape of the exaggerated curve manually the locator needs to be updated to match the new CV positions to have the controller follow along the new curve. By pressing the "Connect to Exaggerated Trail" button the custom function, *connectExagCurve(),* identifies the new CVs of the exaggerated curve and repositions the locator. The locator is then set to follow the orientation of the newly shaped path resulting in the controller following along the new path.

Section 6 allows the animator to adjust the influences between the original path of motion and the exaggerated path using a slider. The custom function, *setCurveBlend()*, is run when the "Change Blend Amount" button is pressed. This function reads in the value set by the trail blend slider and adjusts influences accordingly. By setting the influence to 1 the animator can view a fifty percent split between each curve. Setting the influence to 0 shows the animation along the original path of motion while setting the influence to 2 shows the animation along the exaggerated path of motion.

Section 7 of the exaggeration interface is the graph editor. The graph editor is a graphical representation of the animated attributes within a scene. This allows the animator to adjust the animation curves and the keyframes of their animation. The graph editor within the Stretch-Engine is the same as Maya's built in graph editor. To connect the graph editor to the Exaggeration Interface the MEL functions *getPanel()* and *scriptedPanel()* are used. A scripted panel is one that is predefined in the Maya interface and contains pre-made tools and menus within the panel. The *getPanel()* function identifies the type of scripted panel needed, in this case the type is *graphEditor*, and returns a list of panels with that type. For Maya's graph editor, it is the first object within the list returned by *getPanel()*. The *scriptedPanel()* function is then able to use the panel found and can set it to fit within the Exaggeration Interfaces window. Connecting the graph editor to the interface allows the animator to have all the necessary controls in one location to assist in ease of use when creating squash and stretch. An added feature of the graph editor is that the animator can also adjust the timing of their animation to speed up or slow down the desired animation.

4.4 Roadblocks and Solutions

While developing the Stretch-Engine there were numerous roadblocks that took time to solve. These problems ranged from building a suitable rig system to strange shapes of the resulting motion paths. This section discusses those roadblocks and the solutions created to ensure a functioning tool.

The first roadblock was encountered when developing an appropriate way to stretch the limb structures. When first attempting stretching, there was only one set of joints for each limb segment. As the controller moved in space this initial joint structure would attempt to stretch and match the controller's position. However, this caused an internal calculation issue with the limbs IK system. As a result, Maya would flood the user with warnings and the joints would fail to stretch properly.

To solve this problem, a secondary joint system was created and this joint structure would scale to match the controller. The scaled joint structure would then follow the IK joint structure to properly move the geometry of the selected limb. By creating two joint structures, the IK system would be able to perform its calculations without issue and the second joint structure could scale correctly.

The second roadblock was identifying the keyframes of a selected controller for the timeframe set by the animator. At first, the MEL command *keyframe()* was used to retrieve the position data of the controller. The keyframe command is able to tell the user how many keyframes exist within a section of time specified. It is also able to tell a user which attribute has been keyed during that time frame. However, this function alone was not enough to find the controller's position data.

The online community at stackoverflow.com assisted in identifying the need to list and sort the data returned by the *keyframe()* function for the specified timeframe. The solution to this roadblock is the section of code displayed in the "Paths of Motion" section of this paper. This section of code returns a sorted list of keyframes, from the set start to the end frames, that can later be used to query the position information of the selected controller.

The final roadblock was a strange path of motion that would not follow the torso or neck controller in 3D space. As detailed in earlier sections, a path of motion is made up of CVs that are based on the selected controllers' positions in space. Initially this was done by finding the "world space" position of the selected controller and using this data to draw the new curve. The exaggerated curves would then return the position in "local space" for the selected controller.

World space is the coordinate system for the entire scene. Its origin is at the center of the scene and each position is relative to this point. Local space is the coordinate system from the point of view of an object, meaning the origin point is the object's pivot point. This worked for all controllers except for the torso and neck controller. At first it seemed the problem was because of their positions in the rig structure. They were under the influence of the center of mass controller, causing them to move in space as the center of mass controller moved in space. However, it was due to the controllers' transformations being frozen.

Freezing an object's transformations resets all of its transformation data to zero and sets the object's world space position as its current pivot location in the scene. To fix this problem, a temporary locator would be created that follows the controller during the creation of the original motion path. A locator's transformations are always relative to world space positions and cannot be frozen. This creates an accurate set of position data during the required timeframe. The position data would then be taken from the locator, creating the proper motion path for the selected controller.

5. EVALUATIONS AND RESULTS

The evaluations of this method are conducted by comparing animations based on realistic actions with edited and exaggerated versions. The exaggerated versions are created using the developed prototype tool, the Stretch-Engine. This method aims to improve animator control over the changes in squash and stretch and display that change through motion paths while providing a flexible, user-friendly tool. This section discusses the development of three example animations and how the tool is used to create their exaggerated versions. Each animation is designed to match a physically realistic action from live action videos with the intent of exaggerating a section of the action to create a similar effect to Looney Tunes animations. Videos of the tests and their results can be found at *http://www.zaidhibrahim.com/stretch-engine*.

5.1 Exaggeration of a Single Limb Action

The first test focuses on one part of the body, an arm, to show that the tool can handle a simple change in the limb length. The character is performing a one-two punch combination in this test. The animation is based on a live action video of a professional boxer, Amir Imam, knocking out his opponent, Fernando Angulo [27]. In the created animation, there is only one character performing the punch action. For Test 1, the left arm is stretched to exaggerate the punch.

The animation consists of the character bouncing in place on the balls of its feet; The character's hands are close to its chest, providing cover. Halfway through the animation, the character throws a straight punch with the left hand then a right hook. Both punches

are aimed towards the opponent's head. In Test 1 only the straight punch is exaggerated to change the intensity.

When first creating the exaggerated action, the section containing the straight punch has a scale value of 2, or a 200% change in limb length. During Test 1, the Stretch-Engine is able to generate the original path of motion and the exaggerated path of motion quickly, and the changes applied can be seen clearly. No changes are made to the shape of the exaggerated path and the easing feature is turned off. The result of this simplified exaggerated path of motion is a more intense punch with a change in limb length equal to twice the original length during the final position of the punch.

When the arm is scaled, all dimensions of motion during this action also scale gradually as the arm moves closer to the final position. Because of this, the characters lead up to the final position is also scaled. This scaling is most obvious in the Y dimensions of the controller's position. If the desired action has a lead up similar to the original action but ended in a stretched position, the exaggerated curve needs to avoid scaling the Y dimension. The animator is able to do this through the Exaggeration Interface and can turn off any axis to remove stretching in the specified dimension.

The lead up also changes because the stretch switch is turned on during the exaggerated action. When activated the amount of stretch is based on the ratio values calculated when building the exaggerated path. This prevents the arm from immediately squashing or stretching during the beginning part of the animation as the ratio is closer to 0; this allows it to gradually transition to a full stretch as the arm completes its action and the ratio turns to 1. To achieve the desired effect, the animator can change the deformation ratio caused

by the stretch switch through Maya's graph editor. By selecting the attribute "Stretch Switch" within the graph editor, the animator can see the keyframes that the attribute affects and what value it has at that keyframe.

Although the change in scale applied during the Simple Approach increases the intensity of the punch, a 200% change in limb length does not create a change in length similar to the example animation from Looney Tunes. The example from Looney Tunes is the punch from Daffy Duck in "To Duck or Not to Duck" as it makes contact with Elmer Fudd. The deformation of Daffy Duck's arm is larger than two times the arms' original length, based upon methods of measurement described in the previous section. By increasing the scale applied to the arm, the test animation can create a better match to the Looney Tunes example. Based on this and the difference in lead up, a refined animation test is done that applies the change in scale while excluding the animation's Y dimension.

For this Refined Approach, redrawing the exaggerated path of motion without scaling the Y dimension allows the lead up to remain the same while continuing to smoothly lead into the stretch. The created stretch increases the intensity of the punch similar to the simple test but now the stretch does not affect the rest of the animation as strongly as before. The intensity of the action increases even more when using a scale value of 3, or 300% change in limb length. By combining these two features, a new exaggerated path can be created that achieves the larger intensity while keeping the lead up similar to the original animation. Both the original and the exaggerated path of motion reflect the differences in positions of the left arm, and the animator may freely compare the difference between the two.

Each approach, simple and refined, is timed to see how long it takes to either create the exaggerated path or to edit and adjust it. The steps for creating the exaggeration in both approaches include selecting the limb to scale, finding the appropriate time range, and creating the exaggerated path of motion. For Test 1 both approaches take approximately one minute, with the assumption that the animator understands how the Stretch-Engine works. If we exclude the knowledge of scaling the refined path by 300%, the refined approach takes a minute and a half to complete while the animator tests different scales.

In the case of Test 1 the animated action is quite simple, however factors play in based on animators' preference that can make adjusting it more complicated. In terms of this animation, scaling the original path by 200% in all dimensions does not create the desired effect. The Stretch-Engine is able to control how the stretch affected the dimensions of the animation by containing inputs that allow the animator to set which dimensions are scaled. The Stretch-Engine is also able to rebuild the exaggerated path of motion quickly, allowing testing of different scales to find a scale that created the desired intensity. It also provides another point of control through the graph editor by displaying the degree of squash and stretch along the animation through the stretch switch.

Daffy Duck's punch in "To Duck or Not to Duck" is used to compare the exaggeration of the straight punch. The change in scale of Daffy Duck's arm as it hits Elmer Fudd is meant to increase its intensity, creating the effect of a much stronger punch. By increasing the scale of the dummy arm in Test 1 we create a similar change in intensity. Also, the change in deformation of the dummy arm is similar to the deformation in Daffy Ducks arm,

increasing the length without changing the thickness of the limb. Based on these results, Test 1 of the Stretch-Engine achieves the intended goal of the method.

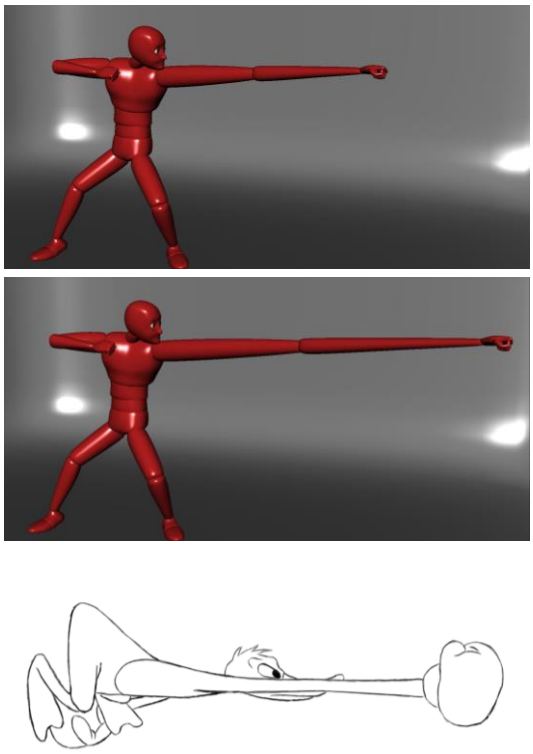

Figure 25: Simple (top) and Refined (middle) test results at the point of impact compared to the example sketch of Daffy Duck (bottom). Sketch based on the episode "To Duck or Not to Duck" released March 6, 1943; Directed by Charles M. Jones.

5.2 Exaggeration of a Pair of Limbs Actions

The second test focuses on both arms of the character to show that the tool can handle changes to a pair of limbs. In this test, the character is swinging a baseball bat as if to hit an incoming pitch. The animation is based on a live action video of home runs and uses the swing stylized by professional baseball player Jose Bautista [28]. In the created animation, the character is holding an object of approximate size to a baseball bat and swings as if hitting a ball. During this test both arms are stretched to exaggerate the swing.

In the reference animation, the character shifts its weight while waiting for the pitch. The character then pulls the bat back and swings as if to hit a ball. After the hit, it looks at where the ball travels while the arms start to relax. To create an exaggerated animation, both arms need to be scaled to emphasize the bat swing. The bat is animated using a constraint that allows it to follow both of the arms' controllers. This constraint is influenced by both arm controllers for the beginning of the animation, but changes influence to only follow the left arm once the hit occurs. This allows the bat's position to update automatically and still follow the arm controllers once the stretch is applied. Although the bat follows along, the limbs need to be holding the bat during the exaggerated action. Similar to Test 1, a simplified approach is used that only scales the limbs and a refined approach that makes additional adjustments; both approaches use a scale value of 2, a 200% change in limb length.

Like the simple approach in Test 1 the arms are stretched by a scale value between 0% and 500%, in this case 200%. The difference in Test 2 is that two limbs are now scaled and these limbs are affecting another object. We first create an exaggerated motion path for each arm and give each motion path a different color to differentiate the curves. The Stretch-Engine is able to generate both of these paths quickly and efficiently. The result is a more exaggerated swing in which there is a change in each arm's length equal to twice that limbs length.

Once the motion paths are created for the simple approach, there is some separation between the hands during the final parts of the motion. This separation is due to the change in scale; as the dimensions are scaled the distance between both hands is also scaled. The

separation can be adjusted in two ways: by scaling the left arm by a percent other than 200 or by using the control spheres to manually adjust the shape of the motion path. Changing the scale requires a few tests to see which scale best fits and can still result in a curve that creates separation, just less than the initial scale. The alternative method of using the control spheres requires fewer iterations, as the animator can see what the new curve shape looks like. They also only need to adjust the keyframes where the separation occurs rather than the entire motion path.

Another concern to address is if the animator likes the scale of the new motion path but feels as if the stretch begins too soon. They can remove a dimension in scaling so less of the curve is affected, but that also affects the final position. In this case they can use the easing feature within the Stretch-Engine. This allows the animator to set a percent of ease that changes the path to either reflect the shape of the original path for a longer period or transition into the exaggerated path more quickly. This causes a more sudden change in squash and stretch compared to the gradual change from start frame to end frame.

In the refined approach, the easing feature is used to create a more sudden effect for this change in exaggeration. By setting an ease-in of 60% the animation has the swing follow the original path for the majority of the selected time frame, and then applies the stretch during the last few frames. The effect is a more sudden change in limb length at the apex of the swing. This effect is applied to both arms to keep a consistent transition between the pair of limbs. Once these changes are made, we use the right-hand exaggerated path as a reference for manually adjusting the left-hand motion path. After manipulating the left-

hand path, both arms hold the bat throughout the animation as it changes to a stretched position before hitting the ball.

Similar to Test 1, each approach in Test 2 is timed to see how long they take. The steps for creating the exaggeration in both approaches include selecting the limb to scale, finding the appropriate time range, and creating the exaggerated path of motion. However, the steps are repeated based on the number of limbs. The refined test also includes additional steps for easing the paths and adjusting the left-hand path manually. For Test 2, the simple approach takes two minutes to complete due to the addition of a second motion path and the steps being repeated to create it. The refined approach takes three and a half minutes to complete. The reason for this increase in time is due to the additional work of manually adjusting the motion path and testing which easing values create the desired effect.

In the simple approach the exaggerated path has proven useful in creating quick and accurate changings in scale. The Stretch-Engine is also able to create paths for the pair of limbs to edit animation simultaneously. In the refined approach we see how the easing feature can be used to control the change in squash and stretch to either slow down or speed up the change in limb length. In this case we used the ease-in feature to slow down the initial transition, speeding up the change in stretch during the final moments of the swing to create a more sudden action.

The simple approach also demonstrates that in cases where the limbs are holding an object, differences in position can cause separation between the limbs when they are scaled. However, in the refined approach we saw the flexibility of the exaggerated motion path as the separation is fixed using the control spheres that are generated along the path. By using

the manual controls, the animator can manipulate either limbs' exaggerated path to create the appropriate shape for their desired action. This new shape can be made to match the second limbs movement or any other movement based on the animator's design.

When comparing the exaggeration of the swing we use a clip of Sylvester the Cat swinging an axe in "Hop, Look Listen." In this clip Sylvester chases Hippety Hopper and tries to hit him. When Sylvester reaches for Hippety or when pulling back to strike his arm stretches. When the animations are compared, the dummy arms stretch in a similar manner to Sylvester's in terms of length. Also, the bat swing with easing reflects the timing of stretch to Sylvester's, as his arm does not stretch throughout the swing but near the end. After reviewing the footage, it can be said that the reason for the change in Sylvester's arm length is the same as the bat test, to increase the intensity of the swing. Based on these results, Test 2 of the Stretch-Engine achieves the intended goal of the method.

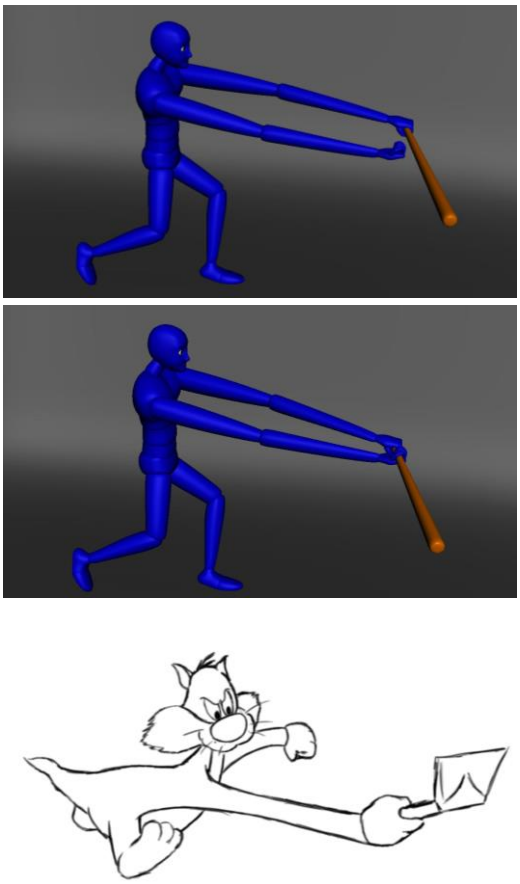

Figure 26: Simple (top) and Refined (middle) results at the apex of the swing compared to the example sketch of Sylvester the Cat (bottom). Sketch based on the episode “Hop, Look and Listen” released April 17, 1948; Directed by Robert McKimson.

5.3 Exaggeration of a Full Body Action

The third test contains a full body action in which the tool focuses on stretching multiple limbs of the body as well as the torso. In this animation test, the character is performing a free running action where it dives over a small ledge, catches itself on the next ledge, and uses its momentum to launch forward and land onto a final platform. The animation is based on a segment of a larger, live action compilation video showing different styles of parkour actions [29]. In the created animation, the character is running over

geometric shapes to represent the ledges and platforms in the real video. To create exaggeration in this animation, the arms and legs are stretched during the first leap while the entire body is stretched during the landing.

The simple approach for Test 3 consists of scaling the limbs during the leap and landing by a scale value of 2, a 200% change in limb length. For the initial leap the arms and legs are stretched. As the character approaches the first ledge, it jumps forward and catches itself on the second ledge. As the character travels between the ledges, it leads its movement with its hands while keeping the rest of the body straight. The legs follow the arc of the body during the jump and then come towards the center of the body as the character catches themselves. For this sequence, an exaggerated motion path is created for each of the characters limbs.

The exaggerated motion path creates squash and stretch by scaling outward from the curve's pivot point. This point is the first keyframe position of the limb found during the range of time set by the animator. This means the curve scales from the starting keyframe. For example, when scaling with the start frame as the pivot, the curve grows outward from the start and the new end position is further past the position at the original end keyframe, creating the change in length. Scaling out from the start frame works well for actions that stretch with the direction of motion. However, if a limb needs to scale opposite the direction of motion then scaling out from the start frame may not create the desired effect.

For the leap section of the animation the legs require a stretch opposite the direction of motion. This type of stretching appears as if the legs are lagging behind the center of mass of the body. To create this effect, the path of motion needs to be stretched from the end

frame towards the start frame. The animator can do this by checking the “Set Pivot to End Frame” box. This reverses the order of the list of keyframes from the selected time line, generating the motion paths with the last frame as the origin point. The result is an original and exaggerated path of motion with the pivot at the end frame and, for this test, a change in scale for the legs of the character away from the body.

With this in mind, the steps for the legs begin by identifying the time range for both legs, identifying the direction of stretch, reversing the pivot point of their exaggerated paths, and creating the exaggerated path for each leg. The paths are assigned different colors to identify the limbs to which they are connected. Before finalizing the leg animation, we move to the arms and repeat the previous steps. Once each limb has been exaggerated they are finalized to update their animation. We then move to the landing and follow the same steps as in the leap, but now include the spine and the neck. The spine and neck are stretched by a scale of 200% and, once each exaggerated path has been completed we finalize the animation.

A scale of 200% increases the exaggeration for both the leap and the landing, however, such a change in scale is too large for the length of distance the character travels. A smaller scale fits more closely, as the limbs are able to stretch but not so far beyond the area the character travels. For example, during the leap the legs should stretch backwards, but no further than where the character starts its jump. As for the arms they should stretch up to the second ledge and not past it.

During the landing the arms, legs, torso, and neck stretch as the character moves towards the lower ledge. In this case, the amount of scaling is also determined by the area

of space available for the character to stretch. The stretch should be contained within the area from the second ledge and end once the character makes contact with the lower ledge. The stretched limb should also follow a path that creates a smooth transition with the non-stretched animation. The shape of the torso should also follow this rule. The middle and bottom torso controllers should be repositioned to better fit the new shape of the spine.

Although the Stretch-Engine is able to create curves for all parts of the body and edit them simultaneously, creating these curves for all limbs of the body at one time obstructs too much of the scene. The scene becomes clustered once the number of paths generated goes beyond three sets. For this reason, it is suggested to use two or three sets of paths to keep the Maya window clean and allow the best environment for editing exaggeration.

These observations, scaling within a confined space and minimizing trails, were considered before creating the refined approach. Since a scale of 200% is too large for the area, a scale value of 1.5, or 150% change in limb length, is used. This scale created a stretch that fits better within the scene while still creating exaggeration. It also contained a smoother transition between the exaggerated poses and the normal animation. When creating these poses, pairs of limbs were adjusted rather than exaggerating the entire body at one time. This allowed for a clearer observation of the changes to the body. Small edits were also made to the spine, the middle and bottom controllers, to better match the new shape of the torso. These adjustments allowed for smoother motion that better incorporated exaggeration that matched the character's environment.

In terms of time, Test 3 takes significantly longer to complete than Test 1 and 2. This is due to the additional exaggeration for the landing. When comparing the leap and landing

as separate animation tests, they take about the same time to complete as Test 2. The simple test of the leap takes two and a half minutes while the simple test of the landing takes three and a half minutes. The refined test for the leap takes around two and a half minutes as well but can be completed faster if all motion paths were generated at the same time. The refined test for the landing takes four and a half minutes to complete with the additional work required to adjust the torso controllers and edit limbs in sets.

Examples of Wile E. Coyote are used to compare the leap and landing to Looney Tunes animation. For the leap we use an action from the episode “Beep Beep.” In this clip Wile is chasing the Road Runner and leaps to grab him. Wile stretches towards the Road Runner but misses as he zooms off screen. When the animations are compared, the dummy arms reach toward the second ledge in a similar manner to Wile and the dummy’s leg stretch emphasizes the action. For the landing we use a scene from “Rushing Roulette.” In this scene, Wile is using spring shoes to increase his speed but almost falls off a cliff. The Road Runner sneaks up and scares Wile as he looks over the edge, and in response he jumps off. As Wile falls off screen his entire body stretches, including his neck and torso. When comparing the animations, the dummy landing creates a similar change in length as it approaches the final platform. Both actions, the leap and landing, stretch to emphasize the actions like the scenes from these episodes. Based on these results Test 3 of the Stretch-Engine achieves the intended goal of the method.

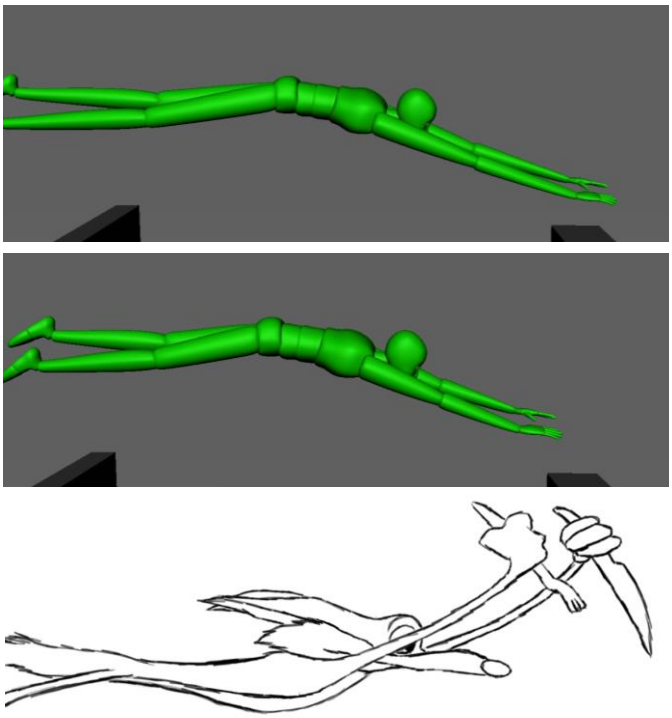

Figure 27: Simple (top) and Refined (middle) results of the leap compared to the example sketch of Wile E. Cayote (bottom). Sketch based on the episode “Beep, Beep” released May 24, 1952; Directed by Charles M. Jones.

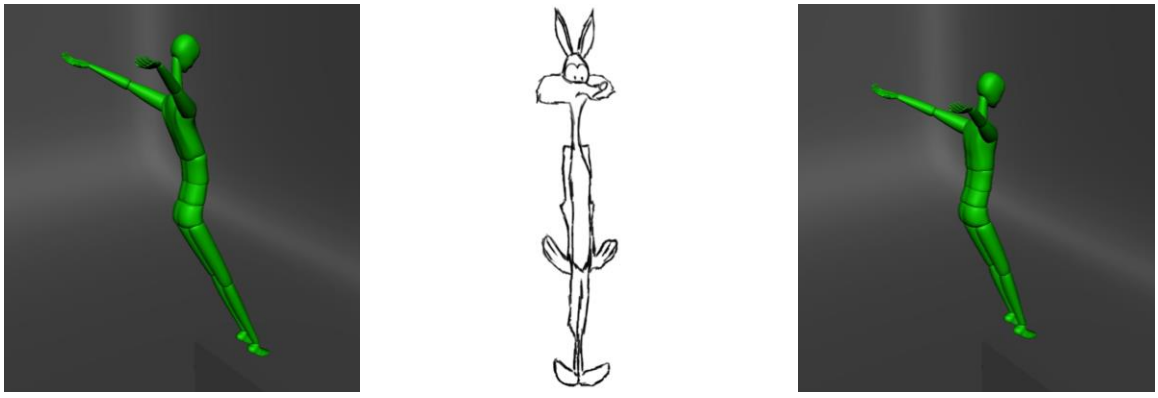

Figure 28: Simple (left) and Refined (right) results of the landing compared to the example sketch of Wile E. Cayote (middle). Sketch based on the episode “Rushing Roulette” released July 31, 1965; Directed by Robert McKimson.

In each of these animation tests, the Stretch-Engine is able to create exaggeration by effectively controlling changes in squash and stretch. The animator is able to create stretching in all limbs of the body while having control on when the stretching starts and ends. The animator can quickly create exaggeration using the exaggerated path and can also make adjustments easily and manually using the curve controllers. The Stretch-Engine tools and functions, such as the easing feature, provide it with an addition level of control to better refine these animations. It can also quickly and effectively produce these paths of motion and can handle single, pairs, or multiple limb adjustments. Through these tests the Stretch-Engine demonstrates the effectiveness of exaggerating motion by squash and stretching limbs through altering paths of motion. The tests also show that an animator can use the Stretch-Engine to create exaggerated motion based on their artistic design.

6. CONCLUSIONS[3]

To summarize, exaggeration is an important part in creating stylized animation. It requires knowledge of the principles of animation, particularly squash and stretch, to achieve a desired result. When creating animations, stylized or realistic, there are a variety of tools that exist to assist animators in their work. Some use physical realistic calculations to simulate animation and others use non-realistic calculations to exaggerate animation by including squash and stretch. Although the tools included in these methods were effective, they lacked certain features that could benefit animators. The physically realistic methods did not have the flexibility to create deformations in a character and when they did create deformations it was only with simple geometry. The non-realistic methods lacked the flexibility to be used within an animator's workflow; they instead overwrote the entire animation to create their exaggerated results.

In conclusion, this paper details a new method to address these issues and assist animators in the creation of exaggerated motion. This method is demonstrated using a prototype software tool called the Stretch-Engine that contains the flexibility to work within an animator's workflow and deform bipedal characters. The prototype focuses on creating exaggeration by giving the artist control over the changes in squash and stretch to scale a character's geometry. The method goals aim towards helping newer artists understand how squash and stretch effects exaggeration. However, it can also be used by more experienced artists, as the prototype was designed to fit within the animator's

[3] Parts of this section are reprinted with permission from " The stretch-engine: a method for adjusting the exaggeration of bipedal characters through squash and stretch" by Zaid H Ibrahim, 2017. In Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '17), Stephen N. Spencer (Ed.). ACM, New York, NY, USA, Article 30, 2 pages. DOI: https://doi.org/10.1145/3099564.3106639

workflow. The changes in squash and stretch are made through the use of generated 3D curves. The Stretch-Engine is written using Python and MEL commands and built within the Maya interface, giving it the capability to be used in an animation process.

The generated 3D curves act as references and controls for the changes in squash and stretch. They reflect the selected limb's current movements to display its path of motion. To create exaggeration this motion path can be scaled, resulting in a scaled change in the selected limbs length, to either squash or stretch the geometry. The scales available are based on a range of scales found by studying the animation style of Looney Tunes animations. This exaggerated motion path also contains controls for manually adjusting the shape of the motion path. By using this exaggerated motion path, the animator can control the degree of squash and stretch they wish to apply to their animation.

Using this exaggerated motion path and other tools within the Stretch-Engine, such as an easing function to slow or speed up changes in scale, animators are able to develop stylized exaggerated motion by deforming the limbs of a bipedal character. When tested, the Stretch-Engine created well-defined exaggerations for animations based on realistic actions and these new versions show similar deformations to examples from Looney Tunes animation. These results show that this method is capable of creating exaggerated versions of motion quickly and efficiently while working with the animator, achieving its goal.

6.1 Future Work

The Stretch-Engine can be a functional basis for future work and improvements because it is written in Python. Python code is flexible and is primarily used when customizing plugins. This flexibility gives the Stretch-Engine the ability to be edited to

work within animation tools other than Maya, such as Autodesk's 3ds Max and Maxon's Cinema4D. There are also multiple extensions, such as a range of style or additional rigs, that can be added to this method in the future to improve exaggeration through changes in squash and stretch.

The Stretch-Engine's range of scale is based on Looney Tunes animation but Looney Tunes is just one of numerous animation styles. By expanding the styles of animation supported by the tool, the animator can benefit from a variety of resources. A "Style Selector" could be created to change the range of scale that the scaled curves are based upon. This slider could include multiple styles of animation, such as art by Tex Avery, Disney Animation and Hayao Miyazaki. When the desired style is selected it adjusts the scale slider to match the minimum and maximum squash and stretch for the new style.

Currently the Stretch-Engine is limited to bipedal characters, but animation contains a variety of interesting characters with multiple limbs. Creating additional rigs for different creatures, such as quadrupeds and insects, for animation can expand the Stretch-Engine's functionality. These rigs could be included within the Stretch-Engine interface and selection of a specific rig would change the tool's limb diagram. This would broaden the range of characters an animator may use instead of limiting the tool to bipedal characters.

The Stretch-Engine provides multiple curves to assist animators in controlling squash and stretch. However, it does not contain the ability to evaluate the animation based on the

camera angle. The perceived intensity of stretch is dependent on the line of action relative to the camera's view. For example, a punch that is stretched and viewed from the side seems more intense than a punch viewed directly in front. This is because in a side view, the camera is highlighting the line of action of the punch. A function could be created to take into account the effect that camera angles have on the resulting animation. This could then develop the best exaggeration curve for the required action based on this camera angle.

The easing function within the Stretch-Engine is currently represented numerically. However, this feature in animation is better represented visually through the use of curves or diagrams. To better highlight this feature a visual representation that can be altered by the animator would better serve in expressing this features functionality. This visual feature would also be easier to understand at first glance compared to testing and viewing the results of the numeric equivalent.

REFERENCES

1. Thomas, Frank and Johnston, Ollie, Disney Animation – The Illusion of Life, Abbeville Press, New York, 1981.

2. Lasseter, John. “Principles of traditional animation applied to 3D computer animation.” *SIGGRAPH* (1987).

3. Chenney, Stephen, Mark Pingel, Rob Iverson and Marcin Szymański. “Simulating cartoon style animation.” *NPAR* (2002).

4. Kwon, Ji-yong and In-Kwon Lee. “The squash-and-stretch filter for character animation.” *SIGGRAPH ASIA Posters* (2009).

5. Roberts, Richard and Byron Mallett. “A pose space for squash and stretch deformation.” *2013 28th International Conference on Image and Vision Computing New Zealand (IVCNZ 2013)* (2013): 166-171.

6. Savoye, Yann. “Stretchable cartoon editing for skeletal captured animations.” *SA '11*(2011).

7. Wang, Jue, Steven M. Drucker, Maneesh Agrawala and Michael F. Cohen. “The cartoon animation filter.” *ACM Trans. Graph.* 25 (2006): 1169-1173.

8. Kwon, Ji-yong and In-Kwon Lee. “Exaggerating Character Motions Using Sub-Joint Hierarchy.” *Comput. Graph. Forum* 27 (2008): 1677-1686.

9. Li, Yin, Michael Gleicher, Ying-Qing Xu and Harry Shum. “Stylizing motion with drawings.” *Symposium on Computer Animation* (2003).

10. Guay, Martin, Rémi Ronfard, Michael Gleicher and Marie-Paule Cani. “Space-time sketching of character animation.” *ACM Trans. Graph.* 34 (2015): 118:1-118:10.

11. Shapiro, Ari and Sung-Hee Lee. “Practical Character Physics for Animators.” *IEEE Computer Graphics and Applications* 31 (2009): 45-55.

12. Hahn, James K.. “Realistic animation of rigid bodies.” *SIGGRAPH* (1988).

13. Bregler, Christoph, Lorie Loeb, Erika Chuang and Hrishi Deshpande. “Turning to the masters: motion capturing cartoons.” *ACM Trans. Graph.* 21 (2002): 399-407.

14. Mordatch, Igor, Jack M. Wang, Emanuel Todorov and Vladlen Koltun. “Animating human lower limbs using contact-invariant optimization.” *ACM Trans. Graph.* 32 (2013): 203:1-203:8.

15. Ansara, Rufino R. and Chris Joslin. “Adding Cartoon-like Motion to Realistic Animations.” *VISIGRAPP* (2017).

16. Cavalier, Stephen. “The world history of animation.” University of California Press, 2011, pp. 160

17. Cavalier, Stephen. “The world history of animation.” University of California Press, 2011, pp. 152

18. Cavalier, Stephen. “The world history of animation.” University of California Press, 2011, pp. 122

19. Furniss, Maureen. “A new history of animation.” Thames & Hudson, 2016, pp. 134

20. Samerdyke, Michael. “Cartoon Carnival: A Critical Guide to the Best Cartoons from Warner Brothers, MGM, Walter Lantz and DePatie-Freleng.” 2013.

21. Furniss, Maureen. “A new history of animation.” Thames & Hudson, 2016, pp. 129

22. Jones, Chuck. “Chuck Amuck: The life and times of an animated cartoonist.” Macmillan, 1999.

23. “MEL for programmers.” *Autodesk Knowledge Network.* 2014. http://download.autodesk.com/global/docs/maya2014/en_us/index.html?url=files/Background_MEL_for_programmers.htm,topicNumber=d30e788742

24. “Python in Maya.” *Autodesk Knowledge Network*. 2017. http://help.autodesk.com/view/MAYAUL/2017/ENU/?guid=GUID-C0F27A50-3DD6-454C-A4D1-9E3C44B3C990

25. Geijtenbeek, T. H. O. M. A. S. “Animating virtual characters using physics-based simulation.” Diss. Utrecht University, 2013.

26. Tan, Jie, Greg Turk, and C. Karen Liu. "Soft body locomotion." *ACM Transactions on Graphics (TOG)* 31.4 (2012): 26.

27. editinKing. “Top 20 Boxing Knockouts of 2015 | Number 16 Amir Imam vs Fernando Angulo.” Online video clip. Youtube, 28 Dec. 2015. https://www.youtube.com/watch?v=JG2OY_7B-gI&t=121s

28. ProSwingNY. “2011 MLB Home Run Derby Slow Motion Baseball Swings” Online video clip. Youtube, 15 Jul. 2011. https://www.youtube.com/watch?v=NrpyBrbu8co

29. Kick-Tube TV. “Most Dangerous Parkour Jumps.” Online video clip. Youtube, 01 Oct. 2014. https://www.youtube.com/watch?v=TQe01rxUisk

APPENDIX A

Python and MEL Scripting

The Python language allows for a flexible coding basis for the Stretch-Engine scripts while MEL provides access to the base commands within Maya. Below are definitions of the functions created using Python and the MEL commands that were used. The definitions for the Python functions are based on their purpose while the MEL definitions are from the MEL command reference in the *Autodesk Knowledge Network.*

*(*http://help.autodesk.com/cloudhelp/2017/ENU/Maya-Tech-Docs/Commands/*)*

Python Functions Created

The follow functions are those written for the functionality of the Stretch-Engine. They are listed in the order that they appear starting with the Stretch-Engine Rig script and then the Exaggeration Interface script.

- distanceCalc() – This function takes two points and calculates their distance in 3D space. It returns a value equal to this distance.
- generateLocs() – This function generates the locators that are used to build the rigs joint structure. Locators are named to match the joints they create and are placed in space along the Dummy model used in the tests.
- buildRig() – This function is used to create the Stretch-Engine rig. It first builds the rigs joint structure, then creates IK constraints along with the stretch functionality, and lastly adds the controllers to the rig structure.

- controlShapes() – The first function within buildRig(), when called it builds a curve based on a predetermined shape. Inputs are pointCircle, feet, hands, crescent, circle, spiral box, box, fourArrows, and lollipop.
- setDKeys() – The second function within buildRig(), it is used to enhance Maya's setDrivenKeyframe() command. It allows the user to place a maximum and minimum value for the connected attributes, creating a range.
- hideLockAtt() – the third function within buildRig(), it takes an object and an attribute (translate, rotate, scale) and locks and hides the x, y and z values.
- resetRig() – This function resets all of the rig controllers to their default attribute values. This places the rig into its default pose.
- createAnimCurve() – This function creates an animation curve within the scene that reflects the selected controllers path of motion. It uses a range of time and keyframe information to generate the curve.
- createExagCurve() – This function creates a scaled version of the original motion curve within the scene. It also creates curve control spheres that are used to edit the shape of the exaggerated curve. The exaggerated path creates a gradual stretch in the selected limb that ends at a length equal to the original length times the scale set by the user.
- connectExagCurve() – This function connects the selected controller to the exaggerated path after it has been adjusted using the curve control spheres.
- setEase() – This function applies the Ease-in and Ease-out values set by the user to the remap. This remap is applied to the ratios used in calculating the exaggerated path. The result is a different transition from original motion to scaled motion.

- setCurveBlend() – This function sets the weighted blend between the original and exaggerated path. The value 0 sets motion the original path while the value 2 sets motion to the exaggerated path.
- finalizeNewMotion() – This function updates the keyframes of the selected object to match the new motion.
- deleteAnimCurve() – This function deletes all generated curves.
- selectBody() – This function highlights the selected limb from the body diagram and updates the Interfaces text field to display the limbs controller. It also removes the highlight from a previously selected limb.
- unselectBody() – This function removes the highlight from a selected body icon if it is selected again and removes the controller name from the Interface text field.

All python functions have several parameters that affect the result of the function. Depending on a parameter's position within a functions parenthesis a parameter will be used differently. The following example shows the layout of a Python Function.

connectExagCurve(leftArm, 10, 25, 2)

This function connects the left arm controller to it's exaggerated path after it has been manually adjusted. The start frame is frame 10 and the end frame is 25, altering the keyframes within this timeframe. The blend value is set to 2, fully connecting the controller to the exaggerated path.

MEL Commands Used

The following are MEL commands that are used with the Python functions. They are listed in the order that they appear within the starting with the Stretch-Engine Rig script and then the Exaggeration Interface script.

- spaceLocator() – The command creates a locator at the specified position in space. By default it is created at (0,0,0).
- xform() – This command can be used query/set any element in a transformation node. It can also be used to query some values that cannot be set directly such as the transformation matrix or the bounding box. It can also set both pivot points to convenient values.
- parent() – This command parents (moves) objects under a new group, removes objects from an existing group, or adds/removes parents.
- curve() – The curve command creates a new curve from a list of control vertices (CVs). A string is returned containing the pathname to the newly created curve. You can create a curve from points either in world space or object (local) space, either with weights or without. You can replace an existing curve by using the "-r/replace" flag. You can append a point to an existing curve by using the "-a/append" flag.
- duplicate() – This command duplicates the given objects. If no objects are given, then the selected list is duplicated.
- scale() – The scale command is used to change the sizes of geometric objects.

- rotate() – The rotate command is used to change the rotation of geometric objects. The rotation values are specified as Euler angles (rx, ry, rz). The values are interpreted based on the current working unit for Angular measurements. Most often this is degrees.
- move() – The move command is used to change the positions of geometric objects. The default behavior, when no objects or flags are passed, is to do an absolute move on each currently selected object in the world space.
- makeIdentity() – This command is a quick way to reset the selected transform and all of its children down to the shape level by the identity transformation. You can also specify which of transform, rotate or scale is applied down from the selected transform. The identity transformation means:
- setAttr() – Sets the value of a dependency node attribute.
- getAttr() – This command returns the value of the named object's attribute. UI units are used where applicable.
- connectAttr() – Connect the attributes of two dependency nodes and return the names of the two connected attributes. The connected attributes must be of compatible types. First argument is the source attribute, second one is the destination.
- addAttr() – This command is used to add a dynamic attribute to a node or nodes. Either the longName or the shortName or both must be specified. If neither a dataType nor an attributeType is specified, a double attribute will be added. The dataType flag can be specified more than once indicating that any of the supplied types will be accepted (logical-or).

- setDrivenKeyframe() – This command sets a driven keyframe. A driven keyframe is similar to a regular keyframe. However, while a standard keyframe always has an x-axis of time in the graph editor, for a drivenkeyframe the user may choose any attribute as the x-axis of the graph editor.
    - For example, you can keyframe the emission of a faucet so that it emits when the faucet handle is rotated. The faucet emission in this example is called the driven attribute. The handle rotation is called the driver.
- setKeyfame() – This command creates keyframes for the specified objects, or the active objects if none are specified on the command line.
- joint() – The joint command is used to create, edit, and query, joints within Maya. If the object is not specified, the currently selected object will be used.
- connectJoint() – This command will connect two skeletons based on the two selected joints. The first selected joint can be made a child of the parent of the second selected joint or a child of the second selected joint, depending on the flags used.
- distanceDimension() – This command is used to create a distance dimension to display the distance between two specified points.
- ikHandle() – The handle command is used to create, edit, and query a handle within Maya. The standard edit (-e) and query (-q) flags are used for edit and query functions. If there are 2 joints selected and neither -startJoint nor -endEffector flags are not specified, then the handle will be created from the selected joints.
- createNode() – This command creates a new node in the dependency graph of the specified type. Used to create multiplyDivide nodes to calculate stretching.

- shadingNode() – This command creates a new node in the dependency graph of the specified type. The shadingNode command classifies any node as a shader, texture light, post process, or utility so that it can be properly organized in the multi-lister. Specifically used to create the Remap for the exaggerated path and a Color Blend node used in calculating stretch.
- pointConstraint() – Constrain an object's position to the position of the target object or to the average position of a number of targets.
- orientConstraint() – Constrain an object's orientation to match the orientation of the target or the average of a number of targets.
- parentConstraint() – Constrain an object's position and rotation so that it behaves as if it were a child of the target object(s). In the case of multiple targets, the overall position and rotation of the constrained object is the weighted average of each target's contribution to its position and rotation.
- poleVectorConstraint() – Constrains the poleVector of an ikRPsolve handle to point at a target object or at the average position of a number of targets.
- expression() – This command describes an expression that belongs to the current scene. The expression is a block of code of unlimited length with a C-like syntax that can perform conversions, mathematical operations, and logical decision making on any numeric attribute(s) in the scene. One expression can read and alter any number of numeric attributes. Theoretically, every expression in a scene can be combined into one long expression, but it is recommended that they are separated for ease of use and editing, as well as efficiency.

- window() – This command creates a new window but leaves it invisible. It is most efficient to add the window's elements and then make it visible with the showWindow command. The window can have an optional menu bar. Also, the title bar and minimize/maximize buttons can be turned on or off. If the title bar is off, then you cannot have minimize or maximize buttons.
- showWindow() – Make a window visible. If no window is specified then the current window (most recently created) is used. See also the *window* command's *vis/visible* flag.
- formLayout() – This command creates a form layout control. A form layout allows absolute and relative positioning of the controls that are its immediate children. Controls have four edges: top, left, bottom and right. There are only two directions that children can be positioned in, right-left and up-down. The attach flags take the direction of an attachment from the argument that names the edge to attach. Any or all edges of a child may be attached.
- rowColumnLayout() – This command creates a rowColumn layout. A rowColumn layout positions children in either a row or column format. A column layout allows you set text alignment, attachments and offsets for each column in the layout. Every member of a column will have the same alignment, attachment and offsets. Likewise, the row allows setting of these attributes for each row in the layout. Every member of a row will have the same attributes. The layout must be either a row or column format.

- paneLayout() – This command creates a pane layout. A pane layout may have any number of children but at any one time only certain children may be visible, as determined by the current layout configuration.
    - For example, a horizontally split pane shows only two children, one on top of the other and a visible separator between the two. The separator may be moved to vary the size of each pane.
- getPanel() – This command returns panel and panel configuration information.
- scriptedPanel() – This command will create an instance of the specified scriptedPanelType. A panel is a collection of UI objects (buttons, fields, graphical views) that are grouped together. A panel can be moved around as a group within the application interface and torn off to exist in its own window. The panel takes care of maintaining the state of its UI when it is relocated or recreated. A scripted panel is a panel that is defined in MEL, with all of the required callbacks available as MEL proc's.

MEL commands use flags as inputs, flags modify how a command works. When using MEL commands with Python, flags are represented by named arguments followed by a value. For example, a flag has a name and is followed by an equal sign (=) then the value for that flag is placed after the equal sign. Also, a flag's position does not affect the result of the MEL command. The following example shows the layout of a MEL command.

floatSliderGrp(label='Scale', field=True, width=300, cw=(1,100), min=0, max=5, value=1, step=0.01)

A slider that returns float values is created and labeled Scale. Following the label is a text field that shows the sliders current value. The slider has a width of 300 pixels and the first column has a width of 100 pixels, this column contains the label and text field. The slider has a minimum value of 0, a maximum of 5 and an initial value of 1. The slider values change in 0.01 increments.

APPENDIX B

Stretch-Engine Installation

For the best performance it is recommended to run the Stretch-Engine within Maya 2015 and newer versions, because script development began with the 2015 version and then moved to later versions as the project developed. A few steps are needed to ensure that the prototype works correctly within Maya's workspace, a read-me file with the steps is included with the folder containing the prototype scripts. The steps for installation are also described here.

1. Go to *http://www.zaidhibrahim.com/stretch-engine*
2. Download the file *stretchEnginePackage*. This folder contains all files needed to install the prototype along with the dummy model.
3. Place the python file "StretchEngineRigScript" within the Maya directory's *scripts* folder. \Users\<username>\Documents\maya\scripts
4. Create a new folder within the Maya directory space and name it stretchEngine. \Users\<username>\Documents\maya\stretchEngine\bodyIcons
5. Copy the entire Python script "ExaggerationInterfaceScript" within Maya's Script Editor.
6. Execute the script in the Script Editor. This will compile all functions of the Stretch-Engine.
7. Once executed the Exaggeration Interface will appear and be ready to use.

The interface script will need to be executed each time you open a new session of Maya. The Script Editor automatically saves its state, or whatever code was in the editor the last time it was opened, and the interface script should still be inside it. The interface resets whenever it is executed but the icons and path creation should connect correctly if the Stretch-Engine Rig or one with the same naming conventions are being used. In case the script is no longer within the Script Editor, it is suggested to save the interface script in your default *scripts* folder and follow the same instructions above to activate the tool.