Rigs of Color

Tool Development: Maya Auto-Rigging Tool

A Major Qualifying Project submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements

for the degrees of Bachelor of Art and Bachelor of Science

in Interactive Media and Game Development

and for the Degree of Bachelor of Science

in Computer Science

Project Team

Terry Deng (IMGA),

Paloma González Gálvez (IMGT/CS),

Patrick Luck (IMGA),

Sophia Marcus (IMGA)

Advisors

Professor Farley Chery

Professor Mark Claypool

Acknowledgements

We would like to thank our advisor, Professor Farley Chery, for being the creator behind the Rigs of Color project and whose input and ideas were necessary to complete this project. We would also like to thank our advisor, Professor Mark Claypool, whose professionalism and expertise allowed us to improve this project immeasurably. We would like to offer special thanks to Laurie Mazza, Tim Marshall and Lisa Liao for laying the groundwork for this very ambitious tool. Finally, we would like to thank all of our test participants who provided invaluable feedback on our project.

Abstract

Hand rigging 3D models is a challenging task. However, existing auto-rigging tools often restrict a user's creative freedom. The goal of this project was to design and implement an auto-rigging tool for Autodesk Maya that allows users to automatically rig bipedal 3D characters, while maintaining a large degree of creative freedom. We created a series of Python 3 systems that create guides which rig the arms, legs, torso and face of the characters, as well as a graphical user interface that connects these systems and integrates the tool into Maya. A user test assessing the time efficiency and usability of the tool found it sped up rigging by a factor of 3.40 times and showed a high user satisfaction rate.

Table Of Contents

1. Introduction

Rigging is a technique used in 3D animation consisting of creating a bone structure for a 3D model (Petty). This set of bones is then manipulated (moved, scaled, or rotated) in order to animate the character. Rigging is an essential process in the entertainment industry because complex 3D characters must be rigged before animations can be created.

Currently, Autodesk Maya is the industry standard rigging software. Maya has a set of rigging tools that allow users to create their own rigs by hand. Hand-rigging is a highly technical skill with a complex learning curve. It is a time consuming process and is largely inaccessible to new users. Additionally, it is a tedious and repetitive procedure that leads to human error even at the hand of experienced riggers.

In order to address the challenges posed by hand-rigging, other developers have created auto-rigging tools that attempt to improve or eliminate the rigging process entirely (Mixamo). Yet these tools often prevent the user from having full creative freedom when creating rigs, providing a series of pre-generated rigs and animations. Additionally, much of this software is inaccessible to many riggers due to steep prices.

The process of hand-rigging is largely complex, tedious and inaccessible to most users, while current auto-rigging tools are costly and eliminate creative freedom. We created a set of Python 3 scripts that provide most riggable elements in a 3D character. We use the PyQT framework to create the user interface for the tool that connects the scripts to the interface. We conducted user testing with riggers of varying experience levels to provide confidence in the usability and efficiency of the tool. The result of this project is an extensive auto-rigging tool developed for use in Autodesk Maya that solves many of the problems found in hand-rigging and other auto-rigging tools.

Chapter 2 provides further background on hand-rigging in Maya, as well as available auto-rigging tools. Chapter 3 provides an overview of the user interface design process; from prototyping to implementation. Chapters 4 through 8 describe the process of creating rigging scripts for the main parts of the body. Chapter 9 details the user testing and evaluation process. Chapter 10 outlines future work and recommendations. Finally, chapter 11 provides the summary and conclusions to the project.

2. Background

The auto-rigging tool MQP was first formulated by Professor Farley Chery. The necessity for the project began through observing a resistance among the Interactive Media and Game Development (IMGD) body from taking on projects that would require high quality 3D models and rigs, limiting game development teams to 2D side scrollers and visual novels (Marcus). These genres are generally lower scope and provide the teams with time to focus on other aspects of their game. A solution needed to be devised that would allow teams to consider 3D rigs and animation to not spend an arduous amount of time creating them.

Autodesk Maya is the game industry standard when it comes to both animation and rigging. This is due to the wide library of tools that are unique and optimized to Maya's workflow. One feature that puts Maya's rigging workflow above many other popular software, like 3D Studio Max and Blender, is the Python 3 integration. Riggers can create Python scripts that can be run within Maya that can call upon the Maya API to do manual actions with the benefit of Python functionality such as for loops, lists, and library features. This allows users to create tools that take actions, like creating a cube and making it red, have Python handle all those actions needed in one button click and remove any manual interaction.

Though Autodesk Maya is the industry standard, Maya rigging is a deep skill that has no universal ways of doing it. There are standard practices and concepts, but the way to accomplish these concepts varies from rigger to rigger. If rigging artists can be thought of as the people who make the puppets for the puppeteers, then one puppet maker may believe 3 strings are enough to manipulate an arm while another insists on using 5. Meanwhile, one artist may prefer to use twine or another may prefer to use cotton strings.

This looseness poses a challenge when creating a general tool because there needs to be some form of cohesion in how different features mesh together, what shape of controller that will be used, what colors should the controllers be, and should the tool make the user select the vertices, edges, or faces in their mesh. It can go even deeper into whether or not the infrastructure should be converted from constraints to matrices. Constraints, despite having a greater depth of knowledge and used mostly in the code infrastructure, can have speed and stability issues. On the other hand, matrix constraints are efficient, newer, and quickly becoming the new industry standard. Agreeing on these aspects allows the tool to feel homogenous rather than a collection of unrelated features that do not fit together.

However, for even the most basic matrix structures, matrices can still contribute more stability and speed because direct connections are always more efficient. The idea to utilize as much of Maya's internal positional and rotational data directly with the use of matrix nodes can go a long way to preventing potential infinite cycling with regular constraints. Simple matrix constraints can always be made with this method and can easily be replicated with Python code. Much more complicated constraints, like parent and aim constraints, require much more advanced knowledge to replicate their behaviors, and can be worth the hassle of recreating to get rid of two memory heavy functions. Flexibility with matrix data can also become a great boon for new rigs. Most notably, the proper use of blending matrix data together can result in a smoother output that the user cannot get with traditional constraints. For instance, a matrix aim constraint can have a multitude of layers that all work together to mimic a real aim constraint. In addition, local transformations on the object being aim constrained can still be reused in other parts of the rig (Marcus). Finally, matrices can prevent any user from accidentally breaking delicate, complicated rigs because matrices work internally within Maya, and thus prevents any possibility of users accidently editing

transformation values. Once further understanding of matrices is achieved, they are invaluable to making any rig setups work smoothly and quickly.

Basic understanding of Inverse Kinematics (IK) and Forward Kinematics (FK) are essential parts of the system. IK provides the basis of motion for some of the tool's features, like the leg, while other features actively allow animators to swap between the two systems. An FK system basically means that in order for an arm joint chain to go from resting at the side of the hip to pointing diagonally away from the body, the user would first need to move and rotate the shoulder joint into place, then the elbow, and then the wrist. FK is similar to how an action figure is posed. On the other hand, IK accomplishes the same task and would be the same as starting from that resting position and simply moving the wrist into position. The elbow and shoulder joints are adjusted as needed.

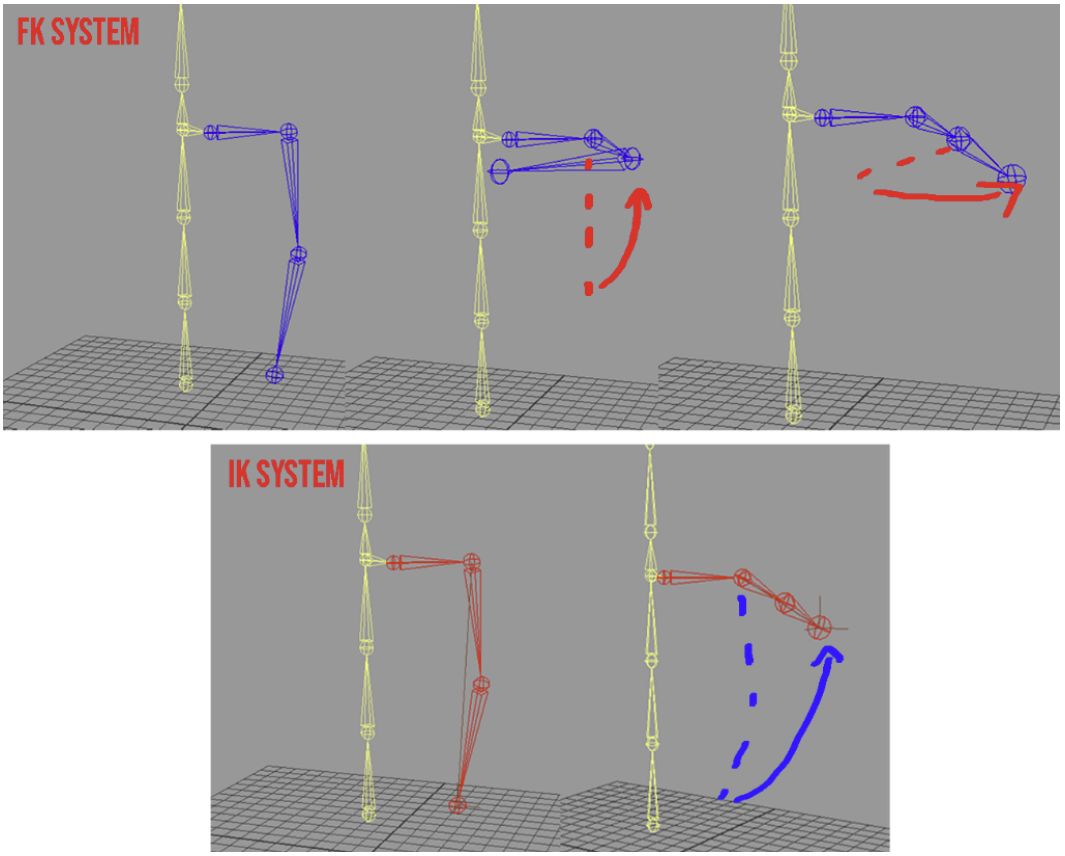


Figure 1. FK systems shows shoulder then elbow, IK system shows wrist moving and the elbow and shoulder moving to compensate

There is also an enhanced variant of the IK system which is another joint system that has a pole vector ride on top of a different IK Handle on the elbow or knee. Acting as almost the grandparent of these joints, it can move them and provide a stability fix to the pole vector. All its data is transferred to a normal IK system through matrix connections. There are pros and cons to both FK and IK systems, but it ultimately comes down to the animator's preference which is important for the tool to provide.

A significant change in Maya based Python rigging occurred when the latest update of Maya, version 2022, implemented Python 3 into its API. This took place mid way through production, but it was essential to upgrade the project to ensure that it was a tool that could be iterated upon in the future. Code had to be reworked to ensure it complied with Python 3. The main issue was analyzing any part of the code that no longer fit with the new Python syntax. Systems that relied on being deprecated in the new version. Some scripts were only minorly affected while scripts that utilized many of these deprecated features were set back significantly. Fully repairing all of the code took several months.

There are several auto-rigging tools in existence that inform the features the tool should have. Other tools either enhance the rigging process with quality of life changes or take out all of the options of the rig artist for full automation.

A notable reference is Mixamo; a well-known and beginner-friendly auto-rigging tool stating that they allow users to "Upload your custom character to Mixamo and get an automatically rigged full human skeleton, custom fit to your model and ready to animate. Customize your rigging options with optimizations for mobile performance" (Mixamo). The focus of Mixamo is to take any model, use their own guide system, and create a rig that only works with their in-house motion capture data library. This allows for people with no rigging or animation experience to create simple animated characters quickly. For those who are

familiar with rigging and animation concepts, Mixamo can become restricting. In order to make edits to existing animations, the user needs to be familiar with animation layers or animation retargeting. Users are also restricted to using the animations provided by Mixamo unless their library resources are not utilized. In this case, the rig will have a mesh bound to a skeleton but missing any controllers or advanced features that the tool provides.

2.1 Remap Value

The use of Maya's remap value nodes comes into all aspects of every individual rig set up this tool creates. Their functionalities are similar to set driven keys, except remaps can take it another step forward in advanced ways with its ability to ratio (Marcus). The node works by taking an inputted value, for example 5, and ratios or "remaps" the input value to a number between an output minimum and an output maximum. The input min and max refer to the acceptable values that the node would take. The graph within the remap attribute page is what the input value will be ratioed by a certain percentage from 0 to 100. This ratio is based on the output min and the output max. All these attributes can be changed freely and do not have to always function as a one to one linear ratio. The value graph can be changed into multiple curves which can go up, down, and even be set to interpolate differently on the graph as a cubic or flat line.

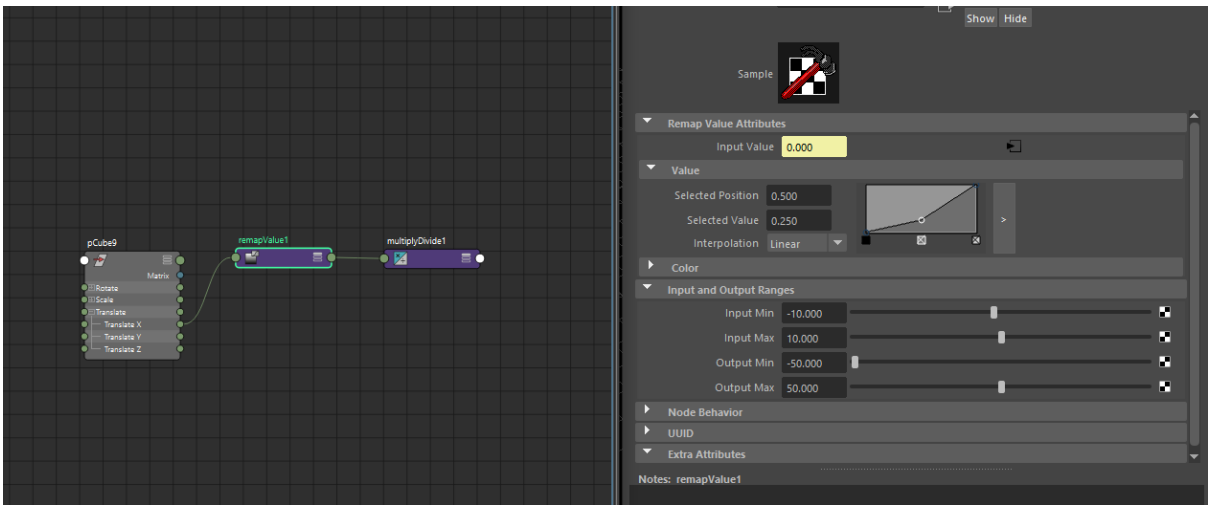


Figure 2 : Remap Value and their settings

The example above is a simplification of the node. To give some perspective as to how this works, imagine another object is feeding an input value. This input value can go infinitely, but the remap node will only accept the input range from -10 and 10. Then, based on the percentage of the input value in relation to its specified maximum and minimum, the value is ratioed to fit properly into the specified output from -50 to 50. In Figure 2, the input value is 0. 0 is 50% between the interval of -10 to 10, or it fits in the position of 0.5 in the graph. So at 50%, it will extract the selected value, or the second ratio 0.25. Finally, the value will change to output 25% between the interval from -50 to 50. The result would give us an output of -25. This simple idea to ratio transformation numbers can be applied in multiple ways that function much more smoothly that a condition node. Additionally, remap value nodes have the advantage of being able to change their conditions and offer much more flexibility than a one time set-driven key. For instance, a multitude of remap nodes can mix their outputs to create a state machine similar to Unreal and Unity state machines (Marcus). These remap nodes can all take the same exact value, but then all have different input and output values that only activate when the input value reaches the specified intervals. Much of the automated systems use multiple nodes to make smoothing cleaner which ultimately leads to a final system less prone to popping or flipping.

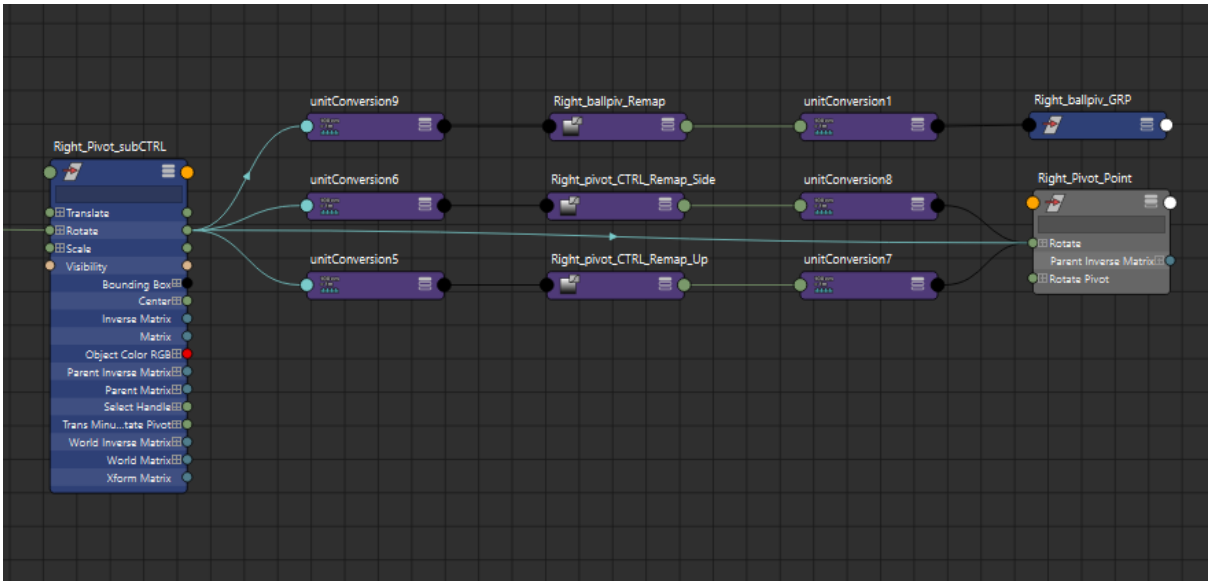


Figure 3 : Example of the remap system for the foot.

A system of remaps to achieve a state machine state for the foot. At certain points using the controller's rotations, the foot will rotate differently based on one of the three remaps: a regular static pivot on the ball, a tilting side pivot, or a tilting up pivot.

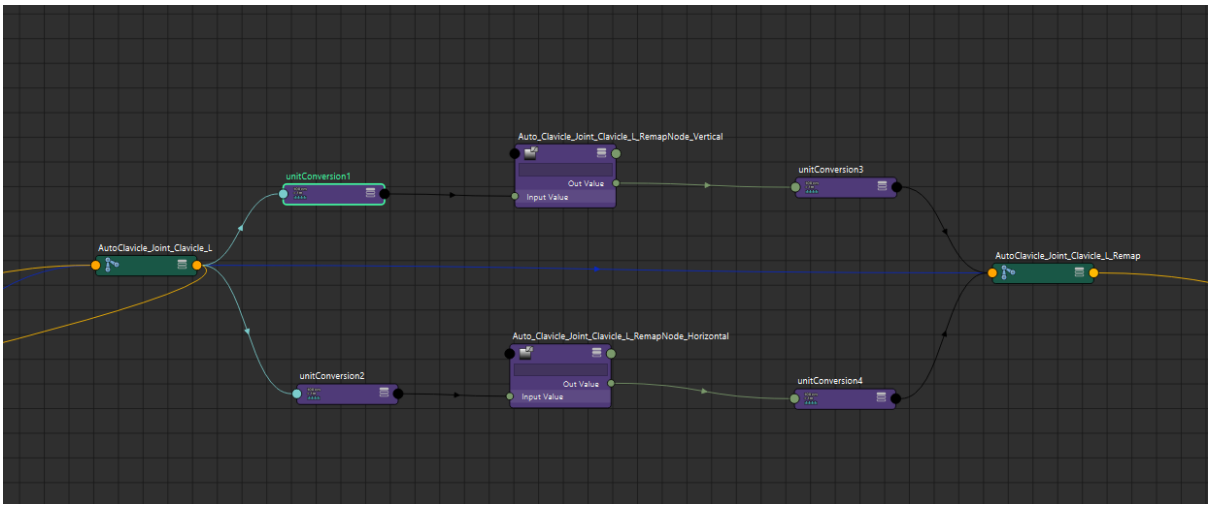


Figure 4 : Example of the auto-clavicle remaps

A simpler state machine of two remap nodes. Taking the translations of the left clavicle, it will either automatically go vertically up when the user raises or lowers the arm, or will automatically move horizontally when the arm is moved forwards and backwards.

2.2 Matrices

Maya matrices were always a part of the Maya foundation for storing values in 3D space, but only recently it was introduced for the public to use with matrix nodes. Its addition into the tech art industry is a great concept for new tech artists to learn because the efficiency, speed, and flexibility of matrices is appealing for anyone looking to create complicated rig systems. Maya matrices and all objects store information in a 4 by 4 grid known as an identity matrix or the default state of the object (Lesterbanks, “Understanding”). The first 3 x 3 area in the top left corner of the identity matrix is where Maya calculates rotational and scale values. With some basic applied matrix math, it is possible to grab a child object’s world position and its local position. To calculate a child object’s world position, the child’s local matrix is multiplied with its parent’s world matrix . To calculate a child object’s local position, the child’s world matrix is multiplied with its parent’s inverse matrix. Using this basic idea, any object is able to grab relevant positional or rotational data. This can be used as a way to create custom constraints, or simply used to replicate the standard Maya constraints already present.

The process behind making matrix constraints is relatively simple. First, the object’s matrices must be extracted and multiplied together with a multiply matrix node. Then, the matrix sum of this node must be decomposed into a usable data format that can be outputted as a translate, rotate, scale, and shear value. Each major attribute correlates to a basic Maya constraint. The “output translate” values can create a point constraint, while the “output rotate” values can create an orient constraint.

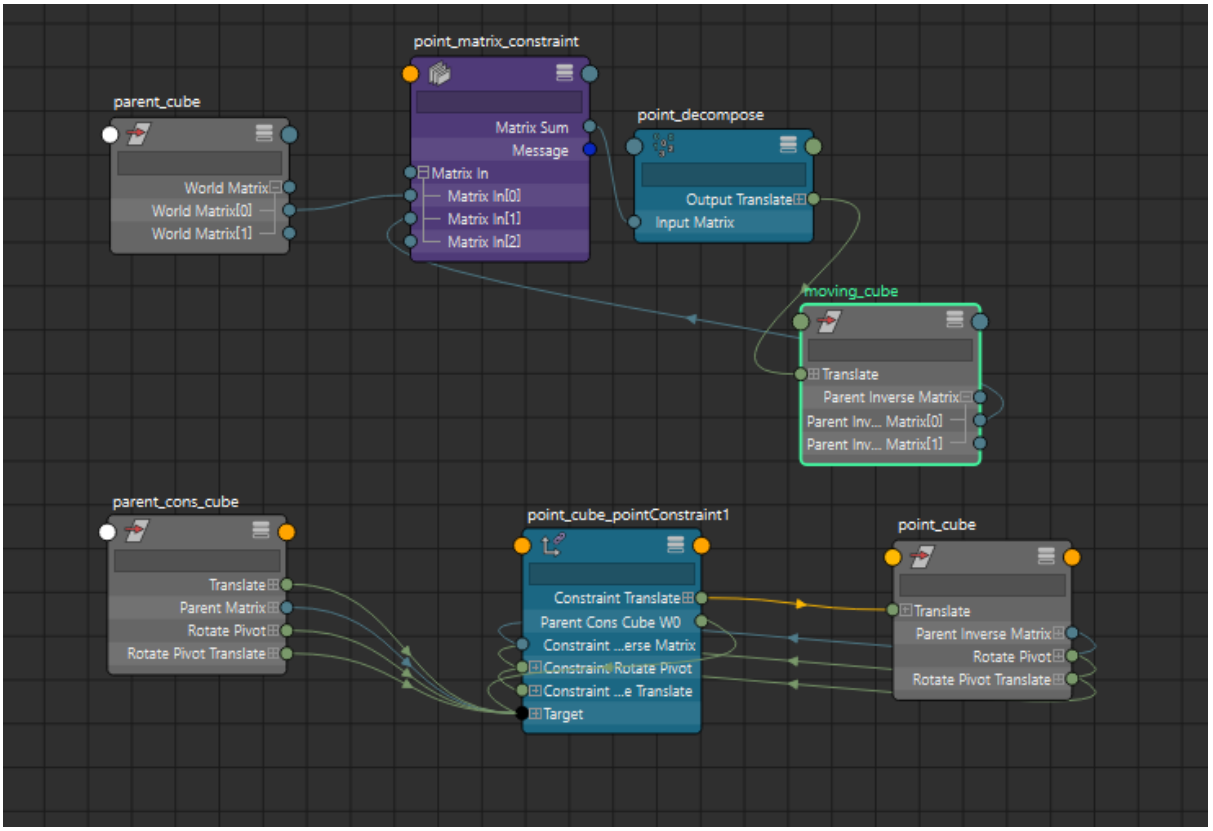


Figure 5 : Difference between a point constraint versus a matrix point constraint

Figure 5 illustrates the potential of matrix nodes and its ability to recreate constraints. The comparison between a simple basic point constraint versus its matrix counterpart is obvious. Not only are there way less connections, there is no risk of the matrix constraint having a chance to infinitely cycle and break.

There are other matrix nodes that are noteworthy for new riggers to understand. The pick matrix node can be used to specify a matrix and only extract information that it is set to. A blend matrix can output values that blend matrix values together with the addition of having an inbuilt pick matrix option (Love, “Custom”).

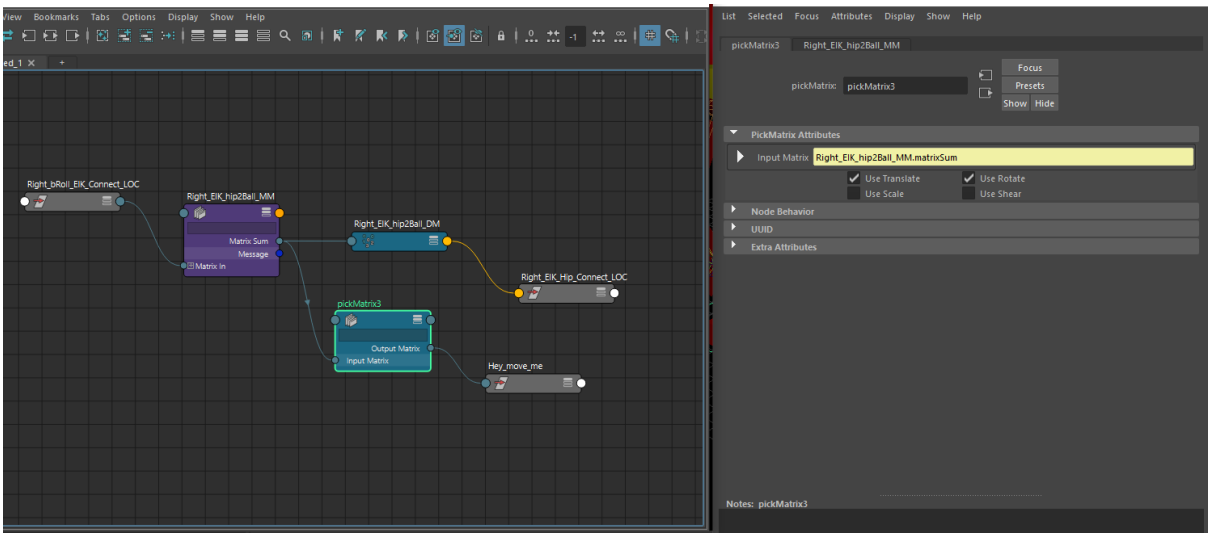


Figure 6 : The pick matrix can pick whatever matrix values to use and apply them.

Lastly, it is very important to note how delicate matrix math can be. For new riggers trying to experiment with matrix nodes, the ordering of adding what matrix goes first or last is to prevent strange math outputs too complicated for a newcomer to understand. Additionally, it is vital that all extracted object matrices must always start from the [0] position to decrease the likelihood of unintentional miscalculations. Finally, the acknowledgement of creating proper offsets will prevent any difficulty in trying to replicate behaviors. While transformation negation, the act of inverting a transformation value to prevent double scaling (Love, “Transform”) through multiplication, can be used with applied matrix math, not applying the offset’s matrix value into a multiply matrix node can still skew results. Using a hold matrix to store offset information is a great way to have a static offset.

3. User Interface

In order to access the rigging tool through Maya, a user interface is created. The interface allows users to interact with the tool and generate all the aspects needed to rig a character. The creation of the user interface followed three distinct stages: prototyping, implementation, and script connection.

During the prototyping stage, the user interface design went through many iterations. Initially, several possible designs were prototyped on paper and tested among members of the team. These paper prototypes were simple and quick to iterate on. When any changes needed to be made due to user feedback, new ones were created almost instantly. Paper prototyping allowed the team to be able to narrow down different interface design possibilities quickly; learning what worked and what didn't for the tool.

Once an initial design had been decided on, a digital, interactive prototype was created. This prototype was designed in Adobe XD, and it allowed the team to test the design of the tool in a more accurate fashion to the final product. The digital prototype features labels for each part of the body the tool can handle, as well as checkboxes that toggle advanced features on and off, and individual rigging buttons for each body part. Additionally, this prototype, as shown in Figure 7, separates the face into its own submenu which would pop open when the "Open Menu" button was clicked.

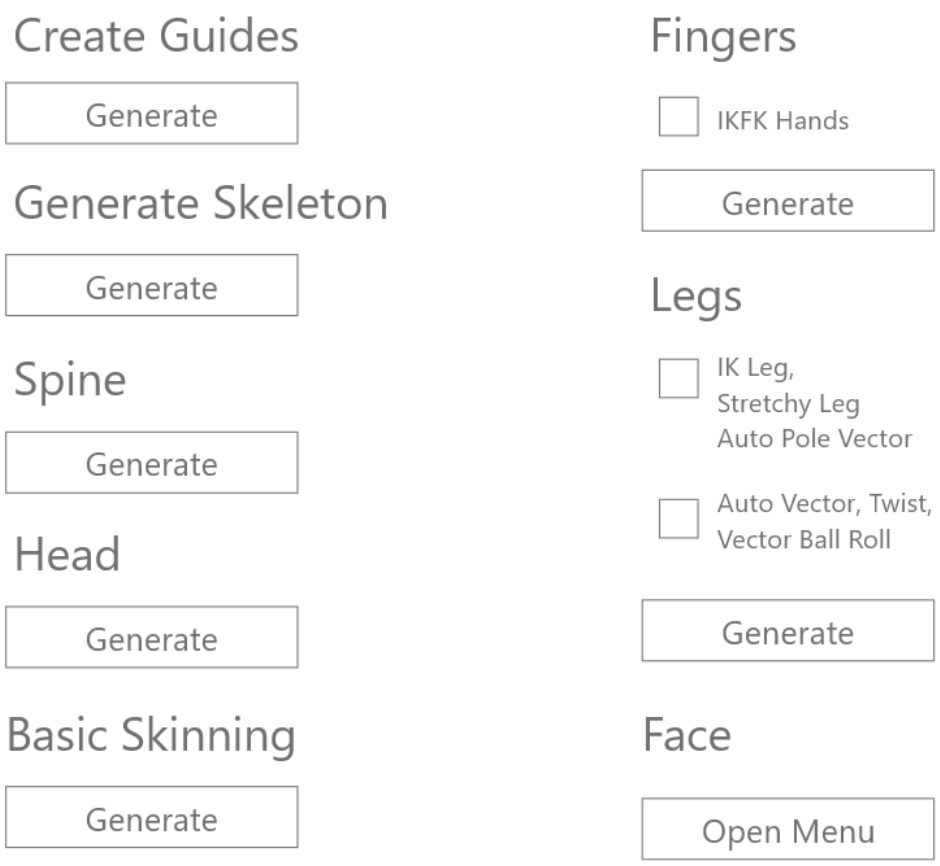


Figure 7. Digital prototype of the User Interface.

While the digital prototype helped finalize the interface, upon further testing and organizing the code, a final design (Figures 8 and 9) was settled on. When looking at some of Maya's native tools and submenus, the UI was divided into two subtools: the head and body. However, instead of creating a separate pop-up menu for the face, a two-tab system was created where one tab houses all of the body's features while the other tab houses the face's. Additionally, the code was restructured in a way where only one rigging function was needed. The interface reflected this change by including only one guide generation and one rig button, which take care of all the body parts that need to be rigged. In terms of the face, a more

complicated set of buttons were used because the face's features require more advanced guides and setup.

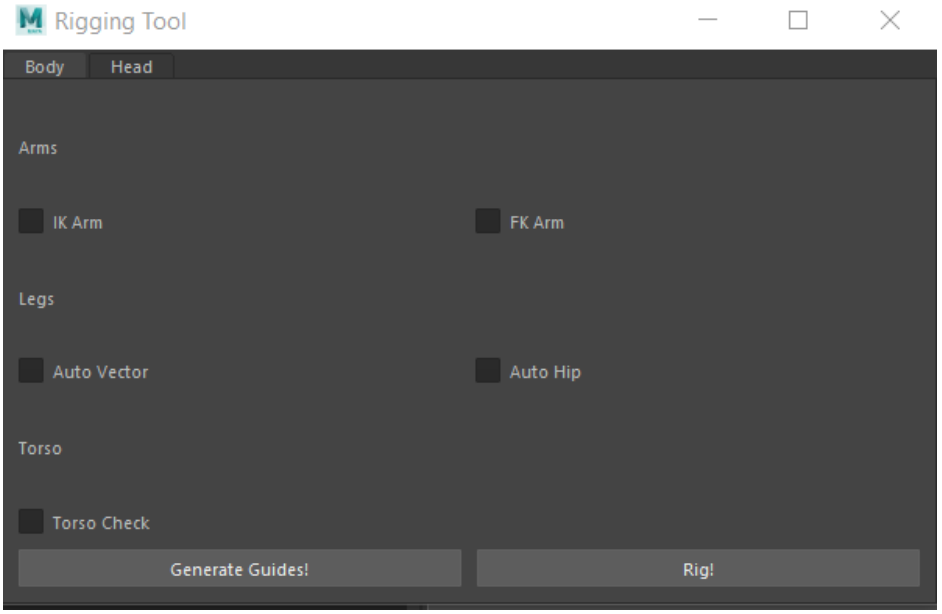


Figure 8. Final design for the User Interface, body subtool.

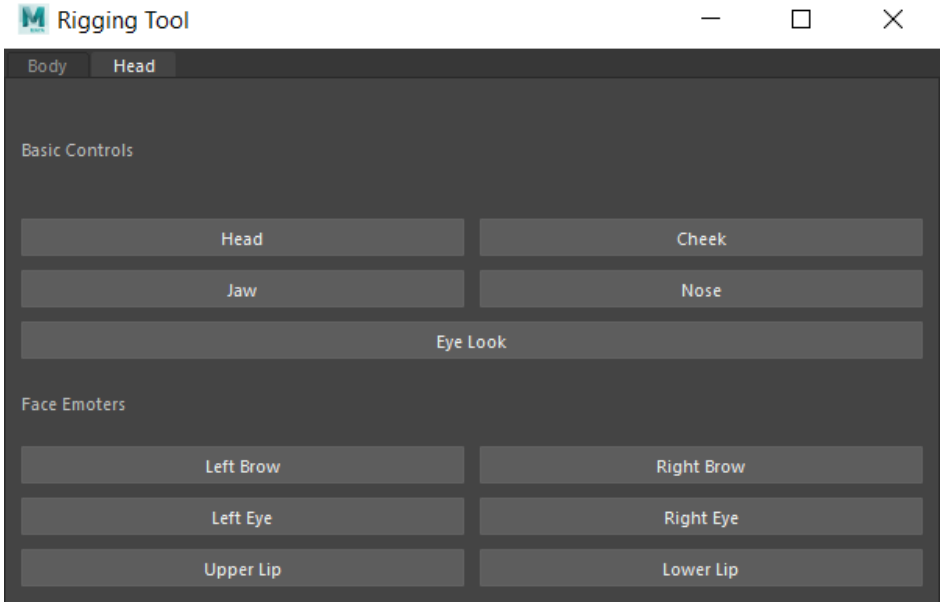


Figure 9. Final design for the User Interface, head subtool.

Once the final design was settled on, implementation began. Specifically using the PyQT framework, the user interface was scripted in Python 3. PyQT is the Python version of the QT framework; a common code library utilized to create user interfaces for interactive projects. Autodesk uses PyQT to develop Maya's native user interface (Zurbrigg). Therefore, in order to stay consistent aesthetically and functionally within Maya, PyQT was used for the tool.

The biggest implementation challenge when working with PyQT was determining how to group buttons, checkboxes and labels into layouts. In QT, a Layout is a set of Widgets (buttons, checkboxes, labels, etc.). There are two types of Layouts used in this interface: HBoxLayout, a type of layout where its widgets sit horizontally next to one another, and VBoxLayout, where the widgets are stacked on top of one another. To create this interface, a combination of both layout types were used. Layouts are also able to contain other layouts and widgets. So typically, a horizontal layout was created with any buttons or features that needed to be side by side, and those were then nested within vertical layouts in each of the two tabs of the interface. Figure 10 provides an example of nested layouts.

Figure 10. Nested layouts within the head subtool.

Finally, once the blank buttons and checkboxes were implemented, the scripts that run the rigging features needed to be connected into the interface. In QT, each button has an attribute `buttonname.clicked.connect(buttonfunction())` that will execute the function `buttonfunction()` as soon as the button is clicked. Since there are two main buttons in the interface, two button functions were created: `createGuides()`, and `createRig()`. On the back end, all of the functions that execute the actual guide generation for each body part are collected in a file. This file was then imported into the user interface file and its functions were able to be called.

In `createGuides()`, the checkbox attributes must first be checked for all three of the body features. This was done by using the QT `isChecked()` attribute for each checkbox and storing each in a variable. Once the checkbox variables were created, they were passed into all of the individual guide generation functions. For example, for the leg guide generation, the function holds three attributes: `mirror`, `autoVector`, and `autoHip`. `Mirror` is a boolean value that simply allows the left and right legs to be generated, while `autoVector` and `autoHip` are checkbox variables. Therefore, when generating the leg guides, the function must be called twice inside `createGuides()`, with opposite values of mirror, as well as passing in the checkbox variables. This process was followed for the arms and the torso and the `createGuides()` function was then linked to the "Create Guides!" button. In contrast, `createRig()` followed a much more simple process. On the back end, there is only one function that needs to be called in order to generate the final rig for all the guides that exist in the Maya scene. This meant that in the `createRig()` function, only one function had to be called once in order to generate the final rig. Finally, `createRig()` was connected to the "Rig!" button.

The last few paragraphs describe how the scripts that create the body rigs were connected to the user interface. Due to extra complexity in setup and lack of time, the face features were not able to be implemented into the user interface. All of the code required to create an advanced face rig has been completed. However, setup needs to be finalized in order for the face guides and rig functions to be ready to be connected to the rest of the tool.

4. The Guide System

In order to set up an automated rig, the tool needs a large amount of information about each different part of the model's body. To do this, a custom guide system was designed that the user can position in the scene manually before the rigging process begins. First, the user moves the various guides into the proper positions along the model's body. Then, the tool creates a joint at the location of each guide to generate a skeleton. As it does this, the guide stores metadata through the use of message attributes into the joints for later use. From there, the automated rigging system uses this information to actually rig the generated skeleton.

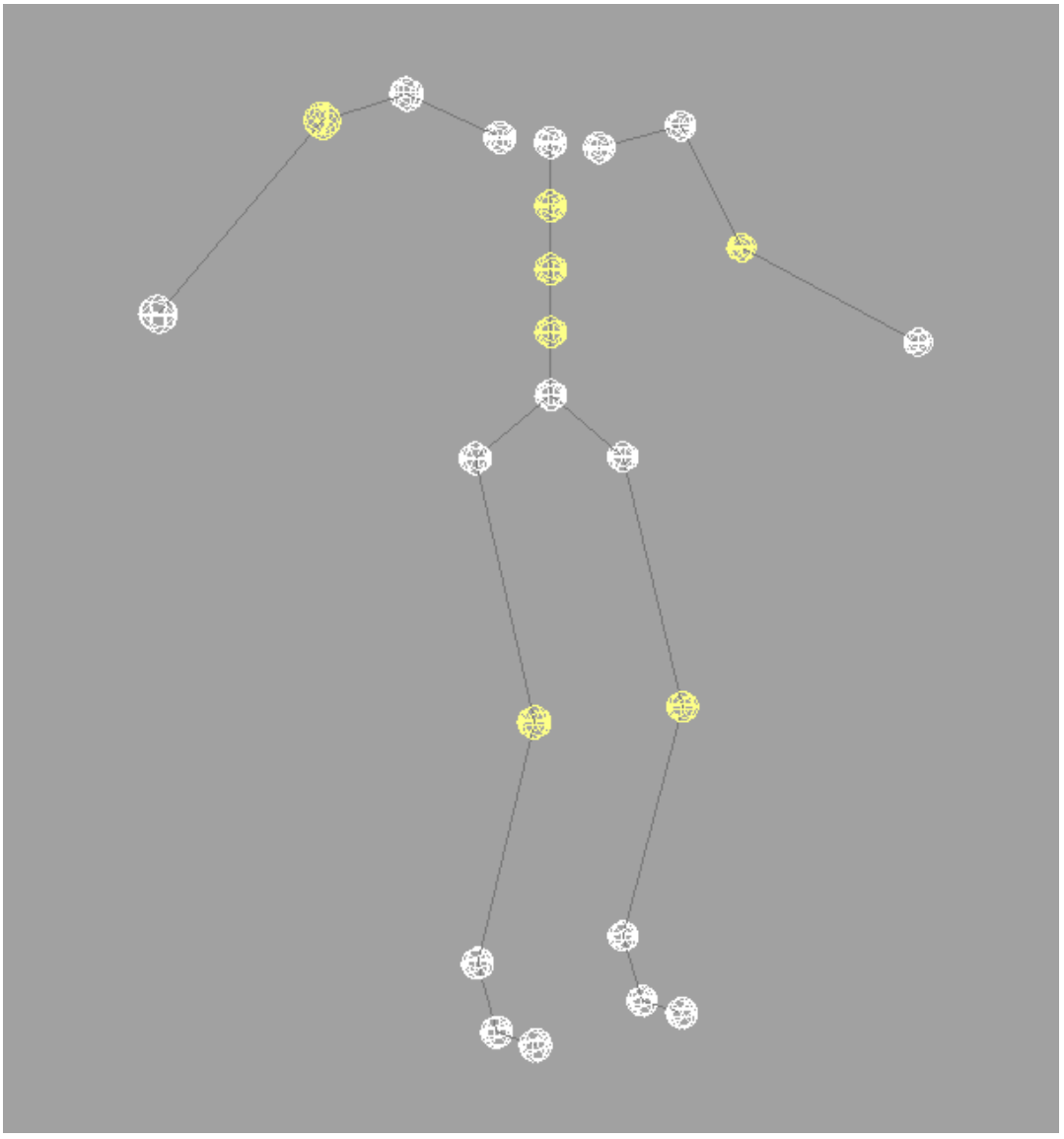


Figure 11; Guide default layout

Each guide serves two purposes: storing positional data and the data of related body parts. Guides are defined in code using a dictionary variable that defines a guide's name, position, and the generated joint's name and parent. The guides also use a message attribute to help identify other important guides in the rigging process. A message attribute is essentially an empty attribute that only exists for the purposes of identifying connected objects. With the message attributes, the rig system can find all of the necessary joints through their guides to find all the joints' relevant information. The benefits of using message attributes is that users do not have easy access to them so they can not accidentally break them, and if the user changes the name of the object in the scene, it is still locatable through the message attribute. Guides are designed to be easy to use for both users and for coders while still being sturdy enough that they won't break easily.

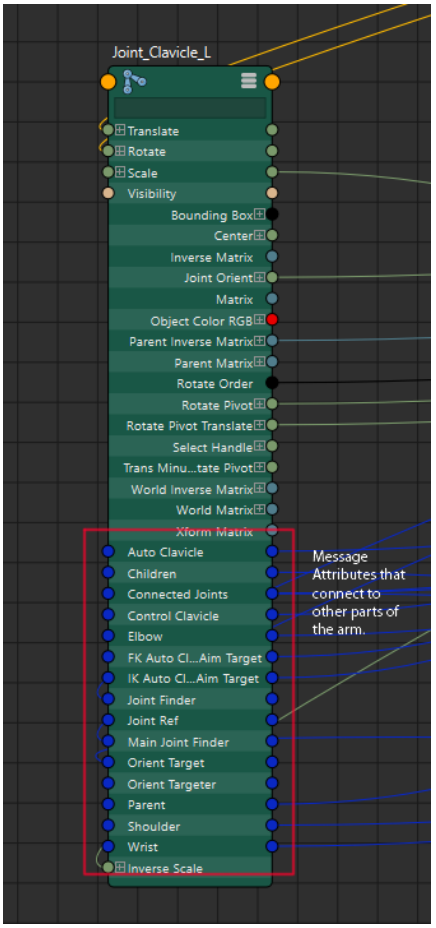


Figure 12; Message attribute system in node editor

Guides are defined in groups; each group being a different limb or segment of the body. They are further defined into three distinct types of guides: base guides, standard guides, and sub-guides. Standard guides are the most common of the bunch and are the simplest to explain. These guides act normally, storing positional information relevant only to the joint it represents. Sub-guides are similar to standard guides except in one way, they move automatically when the user moves a nearby guide. Sub-guides are designed to make the process of setting up the guide skeleton in the scene easier on the user. Finally, base guides act the same as standard guides, but they store a significant amount of extra information in them. Base guides store both the information of the joint they represent, and also various other connections to related guides and joints. Each guide group has one base guide which is used to help the rigging tool find which guides are relevant to its rigging. For example, under the arm guide group, the clavicle is the base guide and connects to the shoulder, elbow, and wrist guides. When rigging begins, the clavicle guide sends a list of all the guides needed for arm rigging to the tool which then uses the information stored in each of the returned guides to perform the rigging process. The way that the system locates the base guides is through an object in the scene called the guide finder. The guide finder is a dummy object whose only purpose is to connect to all base guides through message attributes. From these message attributes, the script can locate the base guides from whom it can then locate all other relevant guides. The guide system is set up so that each guide is easy to locate and necessary information is easy to access throughout the automated rigging process.

5. The Upper Body Structure

The arms are rigged with optional inverse kinematics, forward kinematics, an inverse/forward kinematic switch, automated clavicle, and stretchy arm. The user selects what options they want for the arm before generating the guides, and the tool rigs only what the user wants after the guides are in place.

The inverse kinematic and forward kinematic options are relatively simple in design. First, a separate control skeleton is generated from the skin skeleton. Then, the control skeleton is connected to a controller for whatever form of kinematics the user wants. Finally, the skin skeleton is parent constrained to the control skeleton. In the case that the user wants both inverse and forward kinematics, the system creates two separate control skeletons with one being controlled via forward kinematics and the other via inverse kinematics. The skin skeleton is parent constrained to both of them. Then a switch is created and connected to the two parent constraints so that they can each be switched from one to the other. This allows the user to use either form of kinematics and also be able to switch between them willingly.

The automated clavicle makes it so that the clavicle can be made to move naturally on its own from the shoulder's motion. First, a separate control skeleton identical to the kinematic ones is generated. These control skeletons are not controlled by the clavicle, and thus are independent of its motion. A small target is parented along the length of the shoulder on these control skeletons which the auto clavicle parent group is aimed constraint at. This causes the clavicle to move in the same direction as the shoulder whenever the arm moves. However, this causes unnatural movement, so the motions of the automated clavicle need to be tuned. First, the shoulder is rotated to its extremes by the tool and the rotation of the automated clavicle is recorded for each extreme. These extremes are then fed into a remap node which converts them into more natural values. These remapped values are then fed into

a separate automated clavicle group which actually drives the clavicle, causing the clavicle to move in a natural way whenever the shoulder moves. Finally, this automated clavicle system is hooked into an attribute within the clavicle controller so it too can be turned on and off. The clavicle can also be adjusted manually whether the automated clavicle system is turned on or off, so users still have control over the system. The automated clavicle is designed to make animation easier and more streamlined as it takes care of the important but often overlooked clavicle motion.

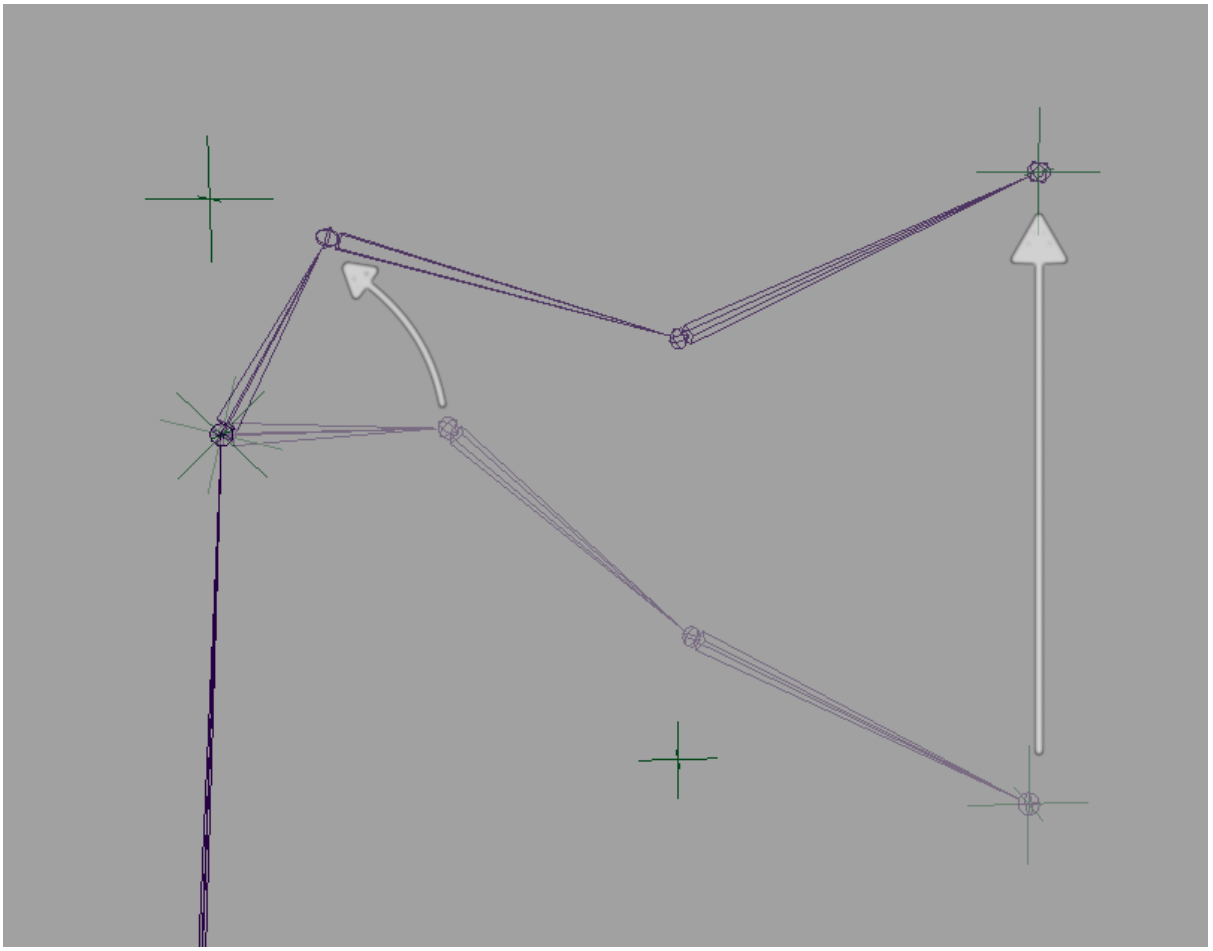


Figure 13; Auto Clavicle reacting to shoulder rotation.

The stretchy arm system makes it so the user can stretch the arm out if they push the controllers beyond what would normally be the arm's length's limit. This is useful for more cartoony animations, or for when an animator needs to better push a motion. The way the stretchy arm works is by first measuring the exact length of the arm. Then, a measurement

tool is created which checks to see what the distance between the shoulder and the arm inverse kinematic controller is. If this distance is less than or equal to the maximum length, nothing happens and it works as normal. But if the length exceeds the maximum length, several things happen. First, the measured distance is divided by the maximum length through the use of a multiply divide node. Then, this value is fed into the x scale of each of the control joints, causing them to lengthen by a ratio such that the arm perfectly matches up with the position of the controller. This makes it so the arm stretches whenever the controller is pushed past the arm's limit, but also makes it so the arm acts as normal under regular circumstances. Finally, the stretchy arm system is plugged into an attribute under the arm's controller so it can be turned on and off. The stretchy arm is a simple system that creates an interesting animation effect.

6. The Lower Body Structure

When approaching the lower body structure the first body part addressed was the leg. Especially in regards to Inverse Kinematic (IK) setups, it is generally one of the first structures riggers learn to rig when learning character rigging . While it is a relatively simple task, it has a wide variety of features that can be added on top of it to make the animation process much easier. The first requirement was to take care of basic features that most animators expected from their IK Leg rig. These features include attribute controlled ball roll and toe tap, stretchy leg, and a vector calculated pole vector.

The vector calculated pole vector is not a common setup that many beginner riggers encounter due to there usually not being a need for it if the leg joints all line up on one axis while facing down another. In most cases, the leg lines up on the x-axis and points down the z-axis. In general the rigger can just put a locator down right in front of the knee and constrain it to the IK Handle on the leg. The problem occurs when the joints have separate coordinates along that x-axis. When attempting to constrain the locator to this askew leg, Maya will try to have the joint chain associated with the IK Handle and form fit to the location of the locator by pointing the central part of the joint chain towards it, taking the rest of the leg chain with it.

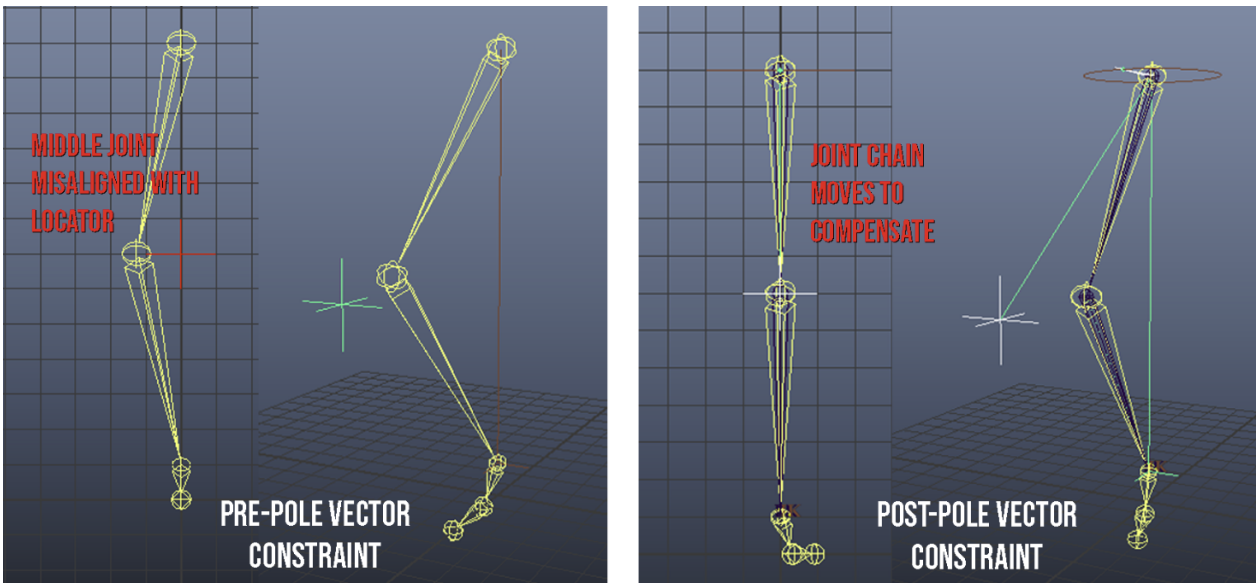


Figure 14. Example of a misaligned leg joint chain compensating to point at the locator during a pole vector constraint between the IK Handle and the Locator

The solution is to utilize the Maya vector system to measure points along the leg in order to figure out where a locator should be placed without the user having to create one. We utilized a video tutorial from rigger Greg Hendrix. He explains that all of the needed functions are provided in the OpenMaya API (Hendrix). By measuring the length of each segment of the leg, it is possible to calculate a spot for the locator to be generated so that the leg remains unaffected and movement with the locator controller goes smoothly (Hendrix).

The Maya vector measurement system ended up being a valuable fix for the stretchy leg system. During production, these systems were being produced alongside each other so utilizing Maya's vector math systems were not the instant go to for the stretchy leg. When creating the stretchy leg it is important to have the measurements of leg, not the distance from the thigh joint to the ankle joint, but the distance from the thigh to knee plus the distance from the knee to the thigh. This gives the actual length of the leg while extended. Typically in a stretchy leg system, users move the leg down until it is fully extended and manually note the leg length from there. Since everything is being handled automatically, without user input

at this point, that luxury is not relevant to the tool; hence the need for measurements. Recording the measurements allows the tool to know when the leg should begin stretching. Without this, the leg begins to appear as if it is just scaling rather than stretching. This is due to the system scaling the joints on the x-axis which is its length-wise scaler by default. Once the leg is fully extended, it gives the illusion that it is stretching, but without knowing the full length of the leg it becomes obvious that the leg is simply scaling.

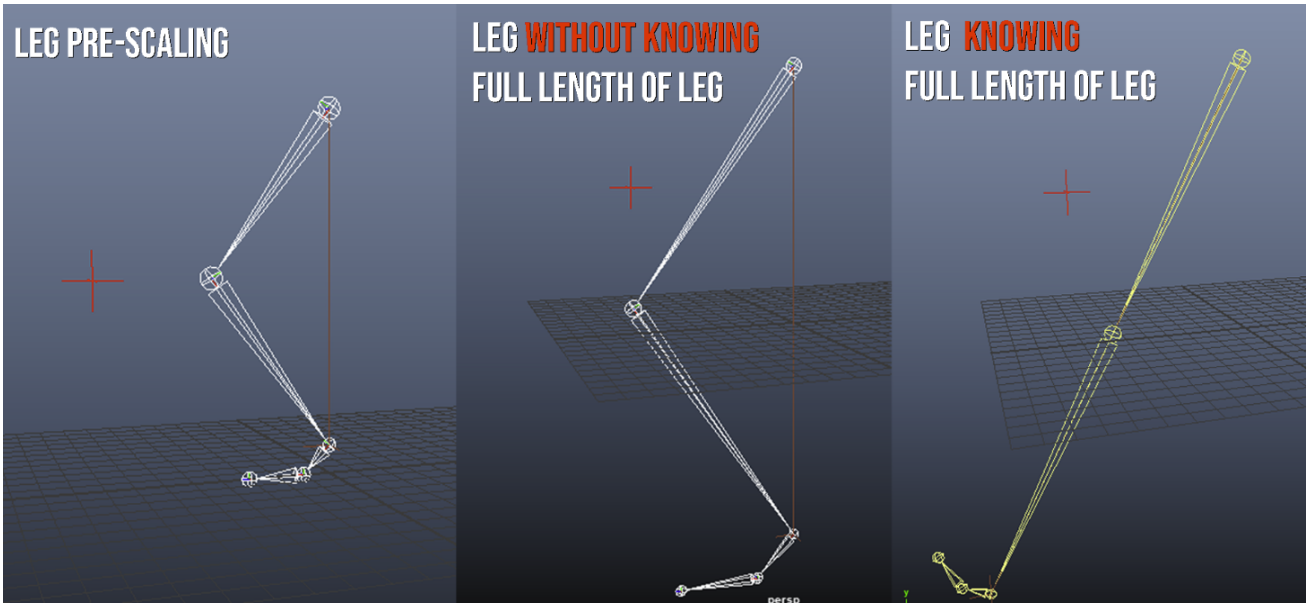


Figure 15. Example of why knowing the full leg length is necessary for stretchy leg

Before discovering the usefulness of OpenMaya's vector math, the tool utilized the built-in Maya distance tool to handle the leg measurements. While an archaic way of handling things by Maya rigging standards, it generally met the needed requirements. The problems took place where the leg would need to be scaled. Distance node tools generate a little line gizmo that constantly measures the point between two spots that the user specifies. Usually, this is via locators that the distance tool is attached to. The issue is that the distance tool only measures in world space even when it is in the local space of another object while parented to it. Since the tool referred to the distance tool to determine the leg's length, if the system was scaled, that fully extended leg length measurement would change due to it being

scaled down in world space. The solution was to find a way to just get a static measurement once and something that the tool could generate at the very start. This is where OpenMaya, a MayaAPI library, came in which allowed the script to take that one measurement and plug it into the system. There were no scaling and measurement issues going forward.

The attribute controlled ball roll and toe tap utilizes remap nodes and was designed by the user crackjack ("Adaptive"). Essentially these nodes allowed the tool to tell parts of the joint chain to move once a certain condition was met. This was mapped to the value of a custom attribute that was attached to the foot controller with a set minimum and maximum value. The toe tap aspect of this feature was maintained, but the team opted for the much more efficient and advanced vector ball roll system.

The advanced vector ball roll is a setting that users can opt into during rig generation. Originally designed by Perry Leijten and readapted to fit with matrices, the foot rig is enhanced to always have some form of contact with the edge of any foot mesh (Lesterbanks. "Check"). In a more simplistic interpretation to describe its functionality, the vector foot roll is a continuously changing pivot that allows tilting and rotation along a Maya NURBS curve. With that changing pivot, a much more complicated remap system is created that can act as the new toe pivot, ball pivot, or heel pivot. Whenever the user rotates their vector controller, it will move the foot's changing pivot's location and switch to function like any of the pivots types with the additional functionality to blend between all of them. This is done with two critical steps: convert 3D rotations into 2D translations, and then use those 2D translations to alter the pivot's new location on a pre-generated oval NURBS curve.

The first step is handled with a vector product of the controller's rotations and a locator, known as the target, that shows us visually which vector direction it is pointing to in space. Essentially, the vector product node converts the 3D nature of the controller's rotational data to become a 2D-output. No other operations are applied because Maya "uses

Matrix math to calculate exact linear position and orientation for points in 3D space (Autodesk, “Matrix”)”. The vector product is a simple way to extract that data. Once connected to a target locator’s translations, users can visually see what kind of output the vector product will give. This can be used as a way to see how a pivot is changing in real time.

The second step is using a curve that takes advantage of the 2D output. The curve is generated with a simple circle that is shaped based on the foot’s length. Having a more symmetrical curve tends to give better results for the side tilting. Then using this curve and the target locator’s translations, another Maya node is used called the nearest point on the curve (nPCI) to allow any object to stick onto the closest point on the curve. Another locator is used for this interaction so users can visually see it matched on the curve. All together, this system gives us a continuously changing locator that moves along a curve.

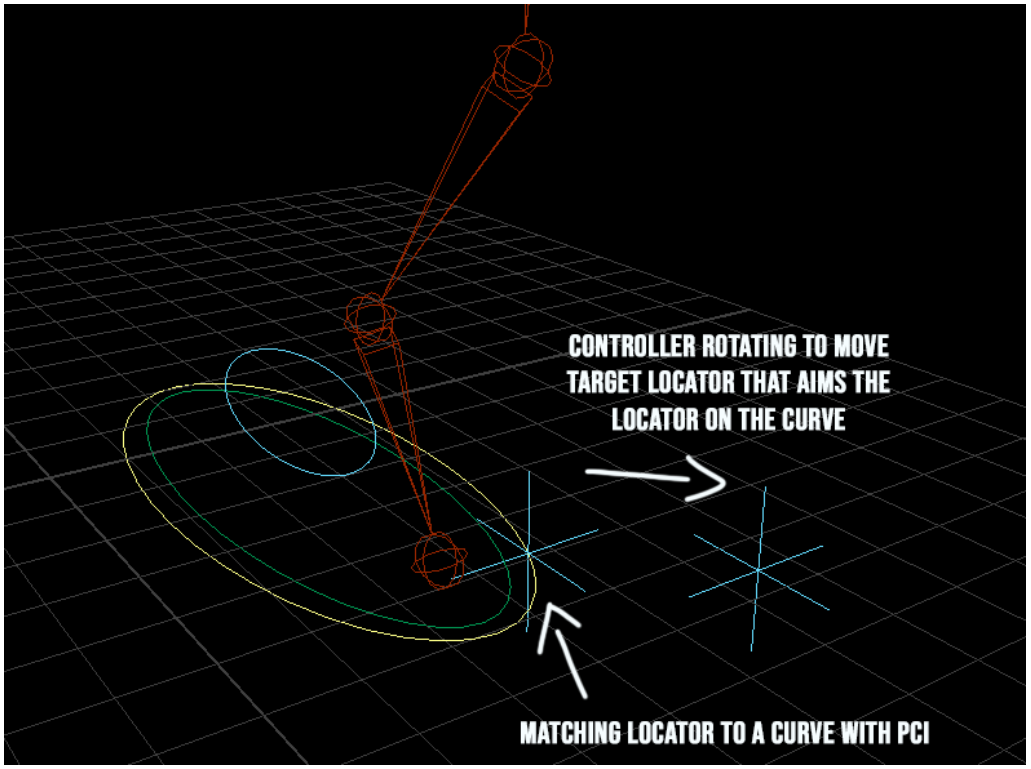


Figure 16. Showcasing the complete vector foot system. Two locators will move an invisible pivot point which is attached to the yellow’s curve shape.

The final step is to connect the locator's position to an actual movable pivot point by using the locator's world location in the form of a matrix. Once this was established, all the toe pivot, ball pivot, and heel pivot groups were parented under the new pivot system.

6.1 Enhanced Inverse Kinematics

After the IK Legs are completed with or without the vector system, the creation of an Enhanced Inverse Kinematics (EIK) is needed for future automation. This enhanced variant of the IK is designed to solve the flaws that a regular IK system has whenever the user attempts to raise a leg in an extremely harsh position. Due to the inconsistencies of a pole vector constraint, regular IK rigs may unintentionally have the pole vector point the knee in the wrong direction. Snapping occurs whenever the pole vector transitions from the front of the knee to behind it. This is all due to the pole vector always aiming the leg's knee joint to itself, but does not know the points where the leg could snap into a positive (in front of the knee) or negative space (behind the knee) unless the user corrects this manually. During the splining process of animation, the pole vector will smooth its translations or rotations from one point to another. A single instance of the pole vector being in the wrong location during interpolation is extremely likely and makes it tedious to fix. The EIK system solves this problem by automatically moving the pole vector to always be in its positive space as the leg moves around. We also begin to utilize Maya matrices a lot more here to improve the speed of automation.

The first part of this system recreates the exact same leg joint structure created by the user. In addition, the tool will also create a two joint structure for our EIK Handle system that is an important foundation for the stability of the IK handles that is added to the copied leg. Using this EIK Handle, we recreate a new pole vector constraint onto this handle, except we

specify how far the EIK's pole vector can automatically go to prevent it from ever snapping and entering a negative space.

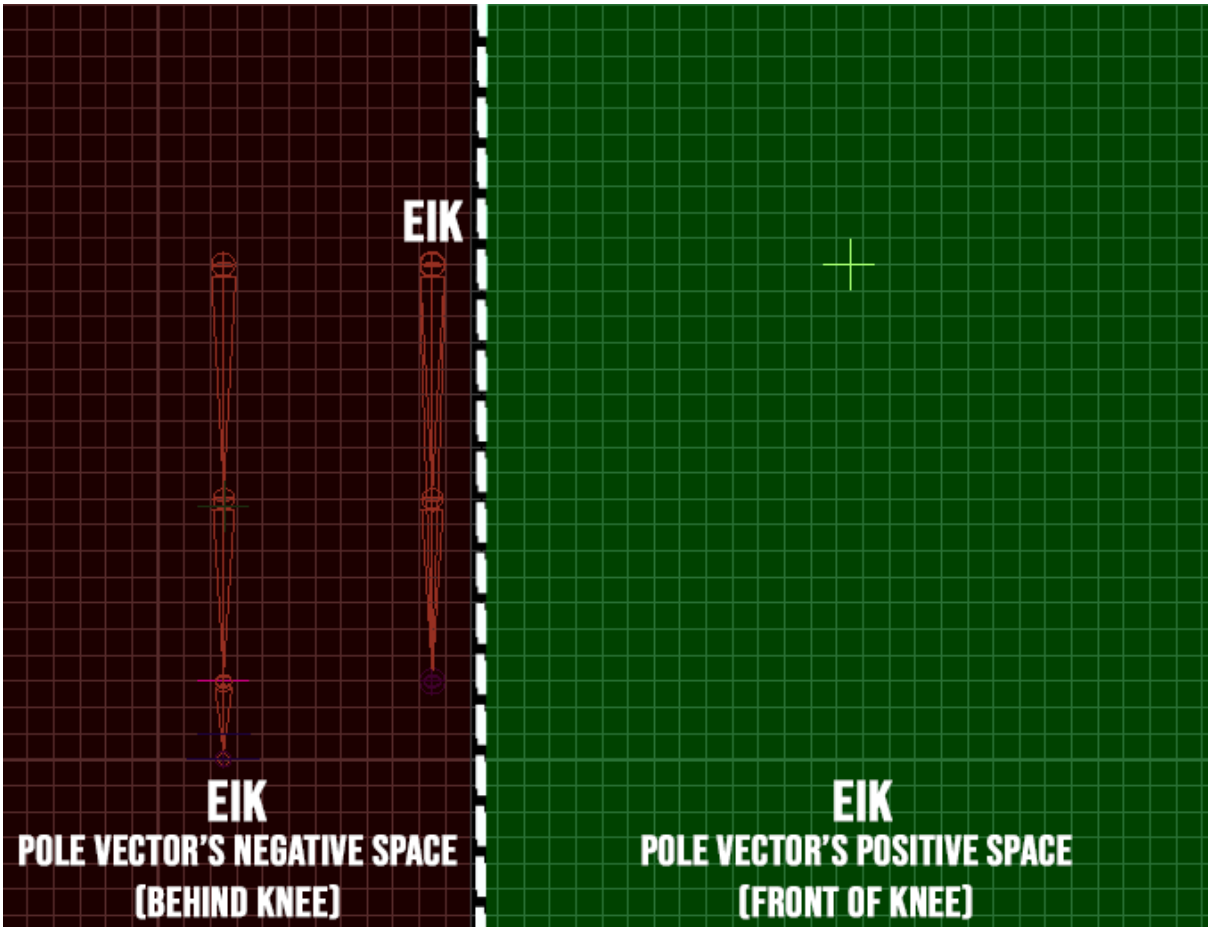


Figure 17. The positive space represented in the green is the area the pole vector is constrained too so it never causes the leg to snap

In the diagram, the dotted line represents the threshold where a pole vector will enter the leg's negative space. A multi-layered aim constraint group always tells this EIK pole vector to always remain in this infinite 2D positive space even while the leg is rotating around. In a 3D context, think of this 2D area becoming an infinite semi-sphere. Another IK handle is also created for the copied leg (referred to as the CIK) so it can later be influenced by this relationship. Figure 18 points to every important layer function and explains what

they do to stabilize the EIK leg, and how it gets connected to the real leg.

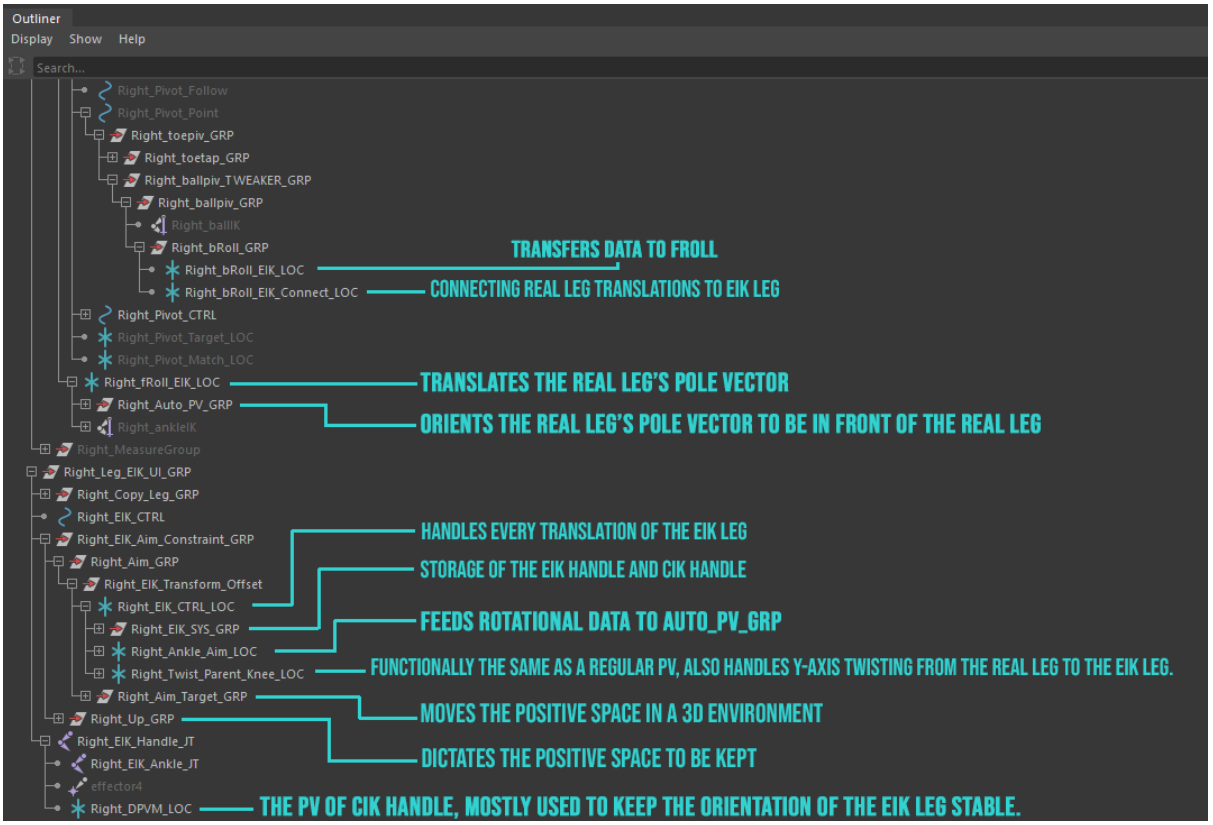


Figure 18. The Hierarchy of the EIK Leg System.

The direct pole vector matcher (DPVM), is the EIK's output we are using as a visual aid to see if the other layers work together to stabilize the EIK leg with a moving pole vector. Once the EIK leg works as intended, the actual leg is then connected to the EIK system so the real leg can drive the EIK leg to move around. All of the calculations end up trying to keep both the ankles of the real leg and EIK leg to move exactly the same based on the ankle's location.

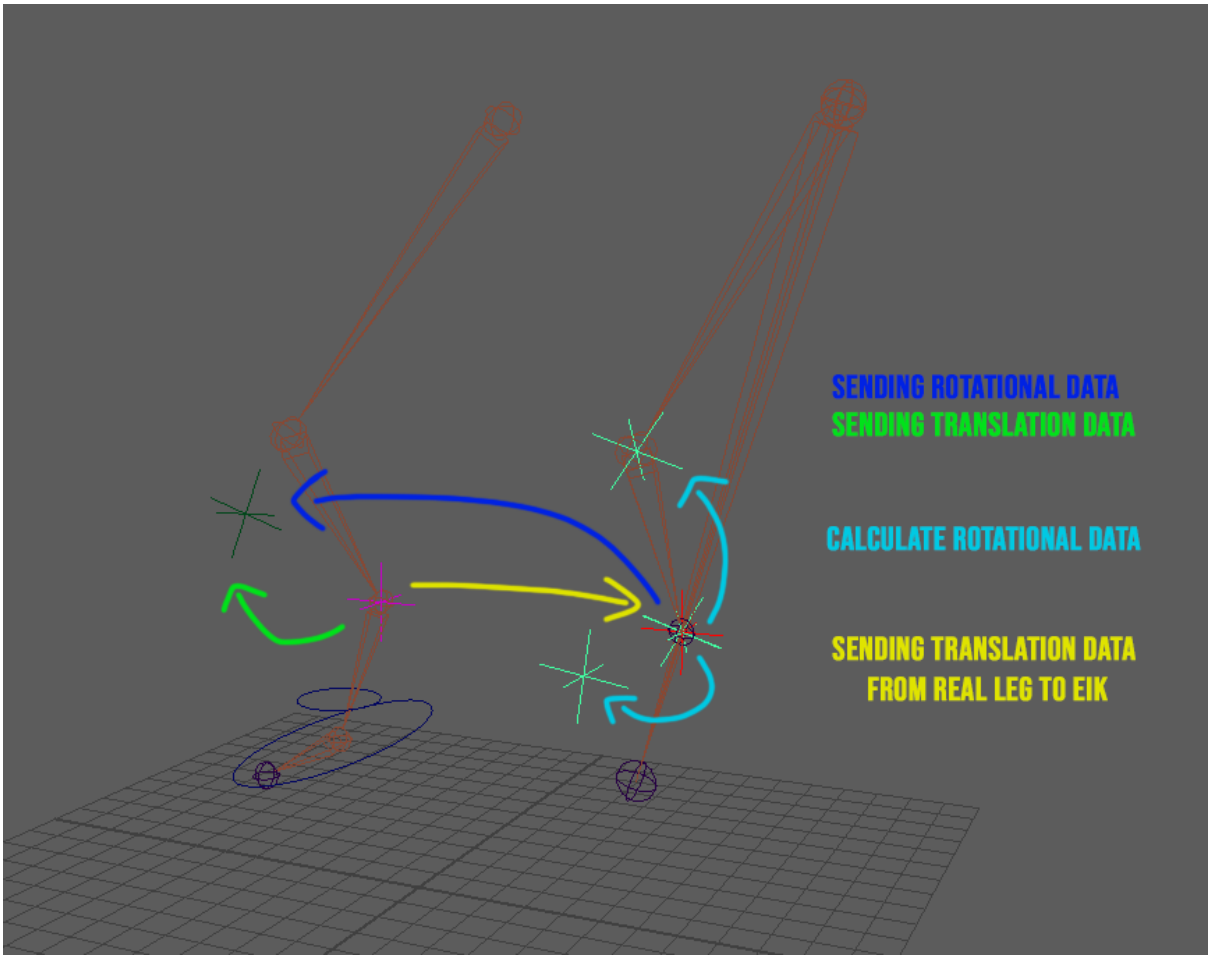


Figure 19. Visual representation on how data is transferred between the two legs. The pink locator feeds data into the red locator which is then fed into the teal locators for calculating the correction rotations necessary to automate the real pole vector.

At the ankle's location is where the automatic pole vector will get the proper positional data from its real leg, and rotational data will come from one of the multi-layered aim constraints. The rotational data is what keeps the pole vector in front of the leg, while the transitional data is what moves the pole around with the leg's position. Even when the user twists or ball rolls the real leg, the multiple layers of the aim constraint group handles each situation through the use of matrices acting as a point or orient constraint to allow rotational data still being fed correctly back to the real leg.

6.2 Auto-Hip and Other Hip Systems

The hip system takes the currently existing EIK systems and adds more functionally to a whole, entire automated lower body. Stability is maintained in a similar way to how the EIK system works. The tool builds a miniature version of the EIK system that automates the hip's rotation in an extremely similar fashion to how the EIK rotates its pole vectors. The only difference is we did not have to account for the pole vector anymore as this is already done with the legs. Instead, this hip EIK only uses a single aim pointer system that is visualized with a cube. The hip EIK grabs the rotational data from the leg EIK to move itself when the leg lifts. Then based on the hip EIK's position, all of its translational data from the hip EIK's knee group is remapped and recalculated to be fed into all the hip's rotations in the Y, X, and Z axis respectively as rotational data.

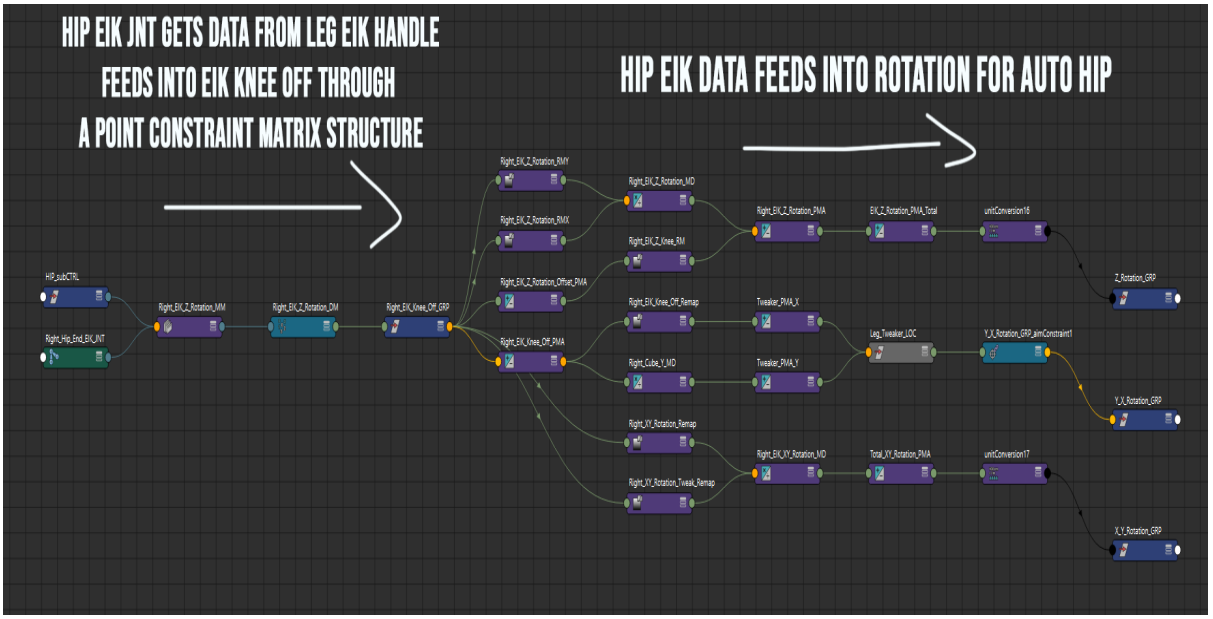


Figure 20. Flowchart of the Hip's EIK way to send automated rotations.

On top of this hip's EIK system, several other minor systems are added and blended into the system. The tweaker controller where a user is able to move the hip into more

extreme up and down positions utilizes the hip EIK system to influence a controllable aim constraint that moves the hip's XY rotations or its ups and downs.

The on and off switch for the auto hips uses a blend color node and matrices to smooth the transitioning of an on and off state which works much better over remapping.

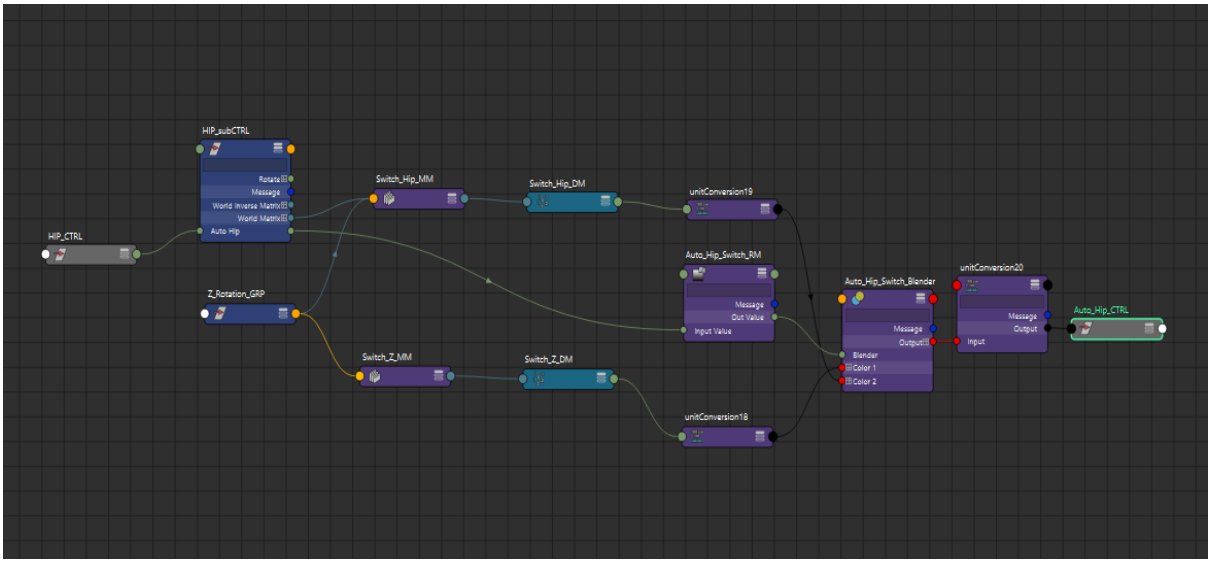


Figure 21. Using matrices with blend colors to create the Auto-Hip Switch

Finally, there is an auto-twist feature for the leg's pole vector that adds the hip's influence to the already existing automatic pole vector that was created earlier. Similarly to all other systems where translational data is converted into rotation, the auto-twist feature does the same process. The tool takes advantage of linear projections from Maya's own matrix system and blends their world matrices together to output a smoother rotation. This process was only recently discovered and managed to solve many problems trying to transfer rotational data from the leg EIK system.

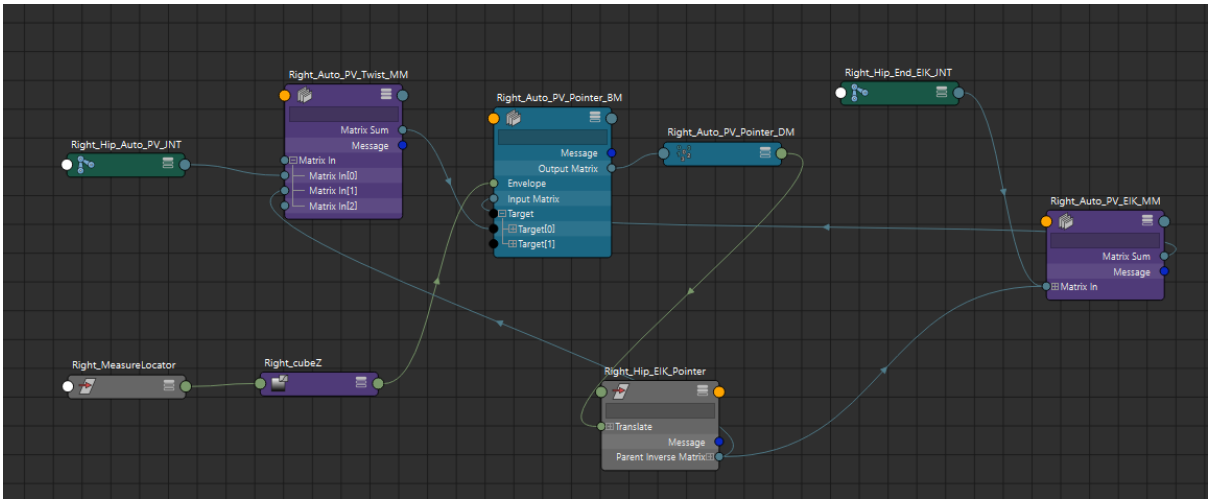


Figure 22. The Blend Matrix Setup

The blend matrix setups allow the pointer to move a certain distance based on the matrix locations of the hip EIK joint and auto pole vector joints. Essentially, there's a feedback loop that changes how the aim pointer will behave. The envelope is a float value from 0 to 1, and functions like a remap node by using much more complicated matrix data. The pointer can then blend between the translate matrices of the auto pole vector joint which is only determined by the cubeZ remap value.

7. The Ribbon

Ribbon splines are used throughout the rig. Ribbon splines are a ribbon-like surface that sits on a series of joints. These joints are pinned to the surface of the ribbon such that whenever the ribbon's shape is deformed, the joints move with the surface. Ribbon splines are useful for twisting, stretching, and bending deformations. The tool generates these ribbon splines automatically, and with the full functionality needed.

The ribbon spline's surface is first created with the number of spans the user wants. Alternatively, the user can create their own surface and plug that into the system to be used instead. Then, surface pins are placed along the length of the ribbon, or in specific locations along the ribbon if the user inputs specific coordinates. Joints are then parented under these pins so that they move with the surface of the ribbon. Finally, the ribbon generates all necessary deformers and control systems the user chooses. These ribbons are then placed along the lengths of the limbs and spine to give proper deformation support to the various parts of the body. The various deformers are set up so that they act independent of one another and stack their motions.

The wire deformer is used to bend and move the ribbon around. The way the wire deformer is generated is as follows: first a NURBS curve is generated along the center of the ribbon, then this NURBS curve is rebuilt to give it 3 control vertices (CVs), and finally a wire deformer is generated on the ribbon with the NURBS curve as the controller. The various CVs are then put into clusters which are parented under controls so that the user can move the wire around to control the ribbon's deformation. The system generates three controllers; one for the top, one for the middle, and one for the bottom. This allows the user to bend limbs and spines in cartoony and stretchy ways.

The twist deformer is used to allow the ribbon to twist along its length. This is done by first creating a duplicate of the ribbon shape. The twist deformer is then applied to this duplicate. Next, the twist deformer handle is connected to attributes within the controllers of the ribbon so that the twisting can be controlled by them. Finally, the duplicate is made into a blend shape of the ribbon so that any deformations done to the duplicate get replicated into the ribbon. Using a blend shape makes it so that even if the ribbon is bent and deformed in various other ways, the twisting will still be along its length. Otherwise, the deformation would not work properly if the ribbon was deformed in other ways. The blend shape, along with the twist deformer handle, is hidden and placed into a hidden ribbon deformer group. The twist deformer allows twisting motions to be easy and natural without requiring much work from the animator.

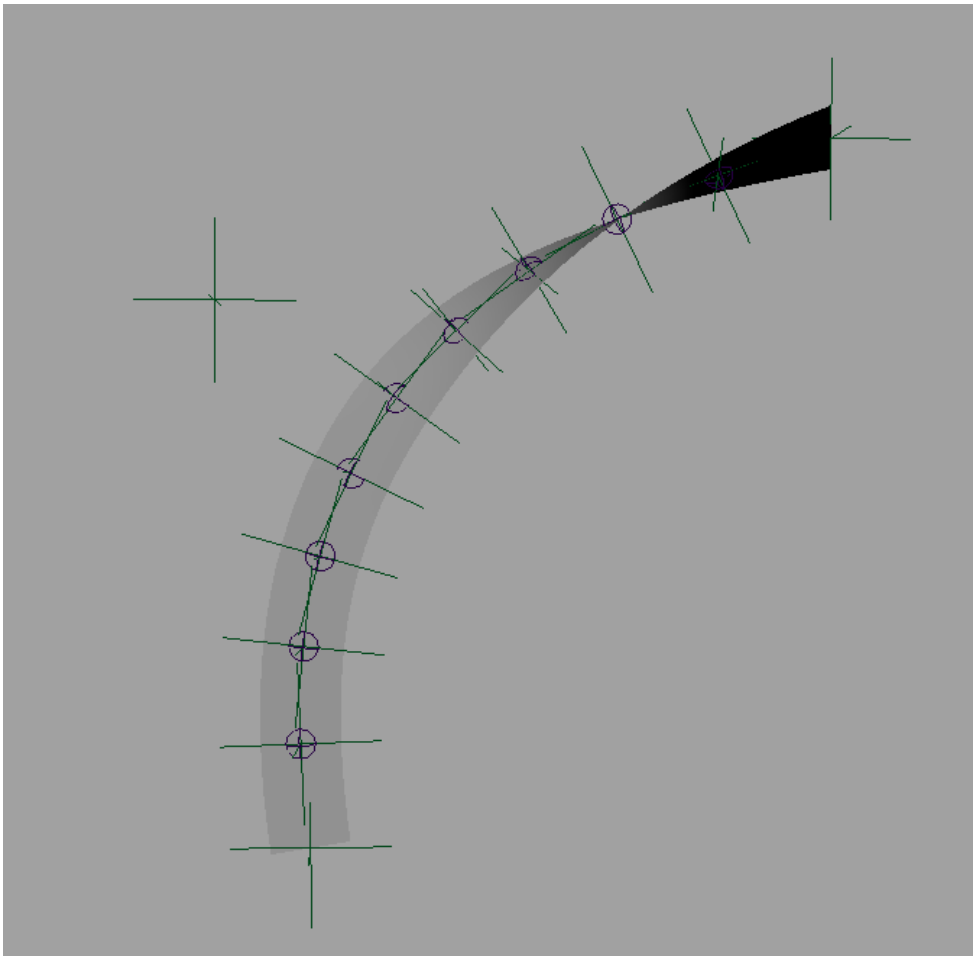


Figure 23; Twist deformer and Wire deformer affecting ribbon.

Finally, the ribbon spline is generated with optional squash and stretch. When an object is squashed or stretched, the inside of the object tends to bulge inward if stretched or outward if squashed. The way the ribbon replicates this is through another deformer. First, another duplicate of the ribbon is created. For each pin on the ribbon, a pair of pins is generated along the length of this duplicate; one in the same spot as the pin on the ribbon, and another along the edge of the ribbon. Next, measure tools are created to measure the distance between the pins along the center of the duplicate and the pins along the edge of the duplicate. Then, a squash deformer is placed on the duplicate, which allows it to bulge inward and outward. When the duplicate bulges, the distance between the pins in the center and their pairs along the edge changes. This new measurement is fed into a multiply divide node as well as their original lengths. This ratio is then fed into a scaling group above the joint. This causes the joints along the ribbon to grow and shrink according to the squashing and stretching of the duplicate. Finally, the squash deformer handle is connected to the ribbon controllers so they drive the deformation. All of these different deformers make it so the ribbon splines allow for a wide variety of freedom for animators.

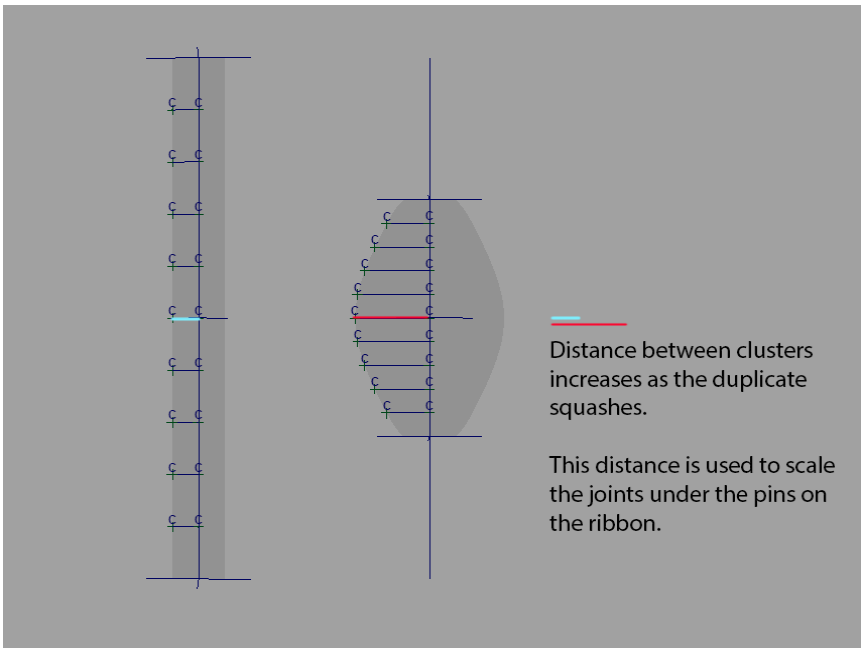


Figure 24; Squash deformer measurement tool example.

8. The Face Structure

The face is a notoriously difficult part of the body to rig due to its complexity and the face typically being the easiest thing for general audiences to critique. Human minds are so used to seeing faces daily. In addition to human nature, there are a multitude of ways to go about rigging a face. When using joints, there is a large amount of eye joints that need to be kept track of, NURBS curves, locators, etc. Creating, naming, and keeping track of all these components is a huge undertaking if done from scratch with no tool assistance.

The tool takes this process and leaves it up to the program to figure out. The main feature that handles this is the Face Emoter system. The system is based on the manual rigging tutorial series created by Marco Giordano that focused on the creation of a cartoon face eye rig (Giordano). This is what creates the foundation for the lid joint and controller system. A big challenge in creating this face emoter system is figuring out what sorts of user inputs are needed to make this an accessible feature while also not trying to automate the entire feature. The latter of which was considered, but would have required the tool to detect nearby vertices and make a decision on how to set up the eye.

The auto detection system ended up being too high scope and would require an egregious amount of testing a multitude of models in order to feel confident enough to ship it. It was decided that the user would choose landmarks of the feature they were trying to rig via vertex selection. Once they were selected, the rest of the setup would be taken care of by the tool. The user is required to use 5 vertices minimum, but the tool does allow them to select an unlimited amount of vertices after the meeting the minimum. This allows for more natural deformation on higher polycount rigs.

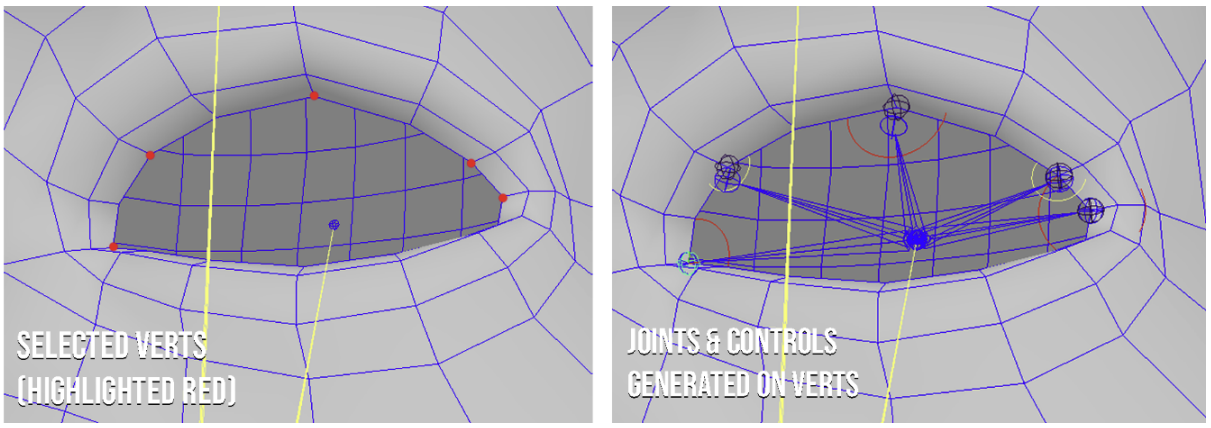


Figure 25. The Face Emoter system being applied to the minimum 5 user selected vertices (verts). Generates the joints and controllers automatically after selection

Interestingly, the process of creating this face emoter system was applied to other parts of the face. Before this point, the feature was only going to be used on the eyelids, but the functionality that the system provided worked well for manipulating parts of the face like the eyebrows and the lips. So long as there was a parent joint to use as a base, the Face Emoter system could be applied to anywhere that had enough polygons to meet the vertex requirement.

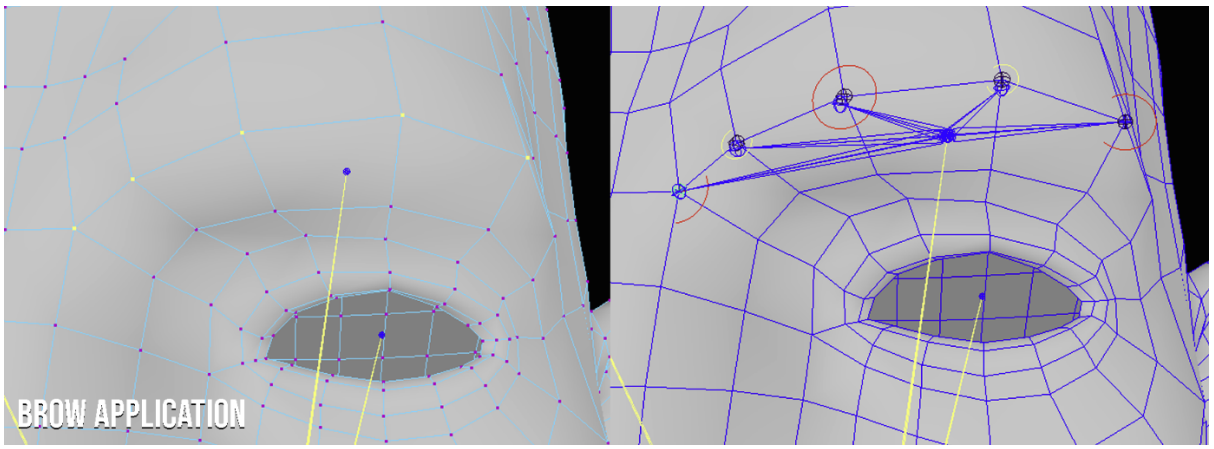


Figure 26. Face Emoter system applied to the brow

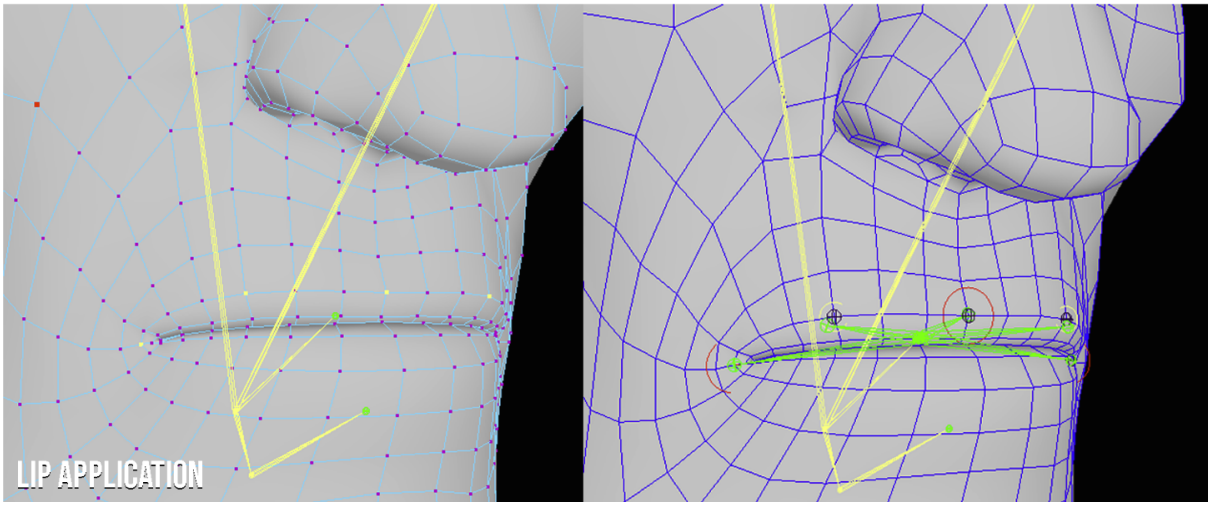


Figure 27. Face Emoter system applied to the upper lip

After settling on how the face emoters were set up, the need for a proper eye look system arose. The system needed a controller that could move the eyeballs freely and also allow for the animator to move both eyes individually should that be needed. This also relied on the existing guide joint system for its positional data and reference to the parent joint. An "EyeLook" joint was created on top of the parent "Eye" joint and parented to it in order to be a part of the skeletal system while also not messing up the Face Emoter side of things. The "EyeLook" joints were then aim constrained to a set of controllers that the tool generated, one for the left and right eye. These controllers are parented under another controller allowing the users to move both eyes uniformly.

9. Evaluation

Testing was needed to ensure that the tool was efficient in the hands of both an experienced and novice rigger. A test that was both stable and relatively beginner friendly was needed in order to test a wide variety of rigging experiences.

9.1 Objectives

IRB approved user testing was held with participants of varying rigging experience levels. The objective of the evaluation was to determine the efficacy and time efficiency of the tool, as well as its usability and user friendliness. Users were asked to perform a rigging test, both with and without the assistance of the tool, and a survey was administered following the test to collect user feedback.

9.2 Procedure

Users were asked to create an Inverse Kinematics (IK) based leg rig within Autodesk Maya 2022. Half of the user pool was asked to begin by performing the manual test, while the other half began by using the tool. After the first test was completed, the first group was provided with the tool while the second group was instructed to rig manually. Both tests were performed using a premade leg model, already loaded into the Maya scene by the team. The tests were timed and the times for each user were recorded.

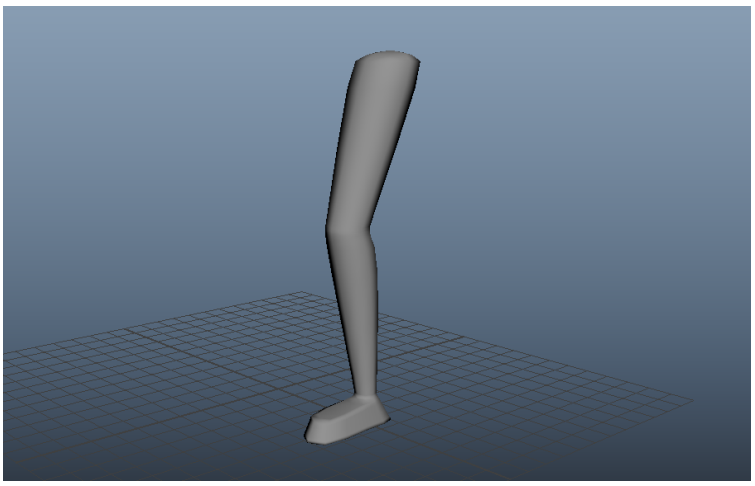


Figure 28. The leg model provided to participants

For the manual test, users were instructed to perform the following procedure:

1. Create a basic leg joint structure that matches the provided leg model. The joint structure must include the following: thigh, knee, ankle, toe, and toe nub. The joints must be named accordingly in Maya.
2. Create an IK tool connection between the thigh joint and the ankle joint.
3. Create a basic NURBS controller.
4. Orient constrain the NURBS controller to the ankle joint.
5. Point constrain the NURBS controller to the IK handle generated in step 2.
6. Create a locator and snap it to the knee joint by holding “V” and dragging it to the knee.
7. Move it out in front of the knee on one axis.
8. Freeze the transformation.
9. Constrain the IK handle to the locator by using the pole vector constraint.
10. Skin the leg with the now created joints.

For the tool test, users were asked to:

1. Open the tool menu.
2. Press the “Generate Guides!” button
3. Line up the guides with the leg mesh provided by moving the gizmos in the scene.
4. Under the “Legs” section, check the “Auto Vector” checkbox.
5. Press the “Rig!” button.
6. Skin the leg with the now created joints.

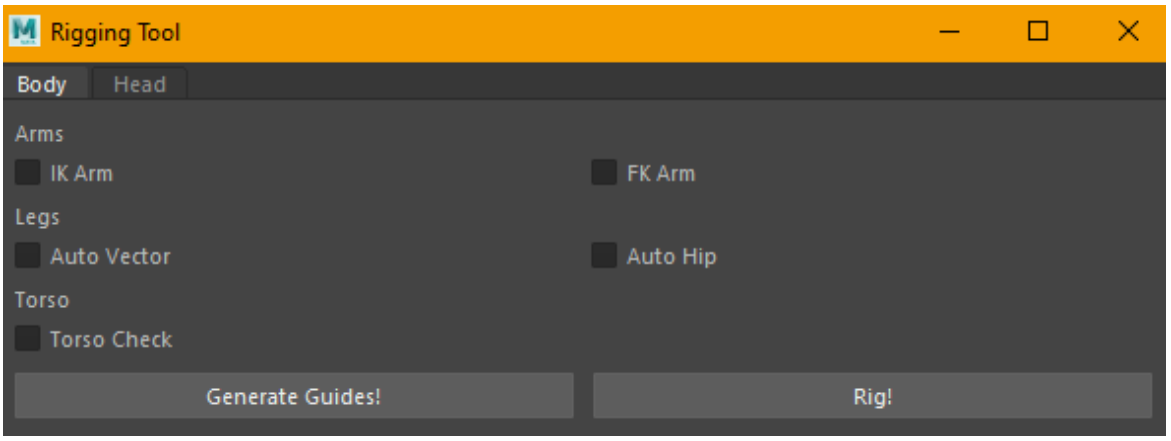


Figure 29. The body tab of the User Interface

While the manual test goes over concepts like creating an IK Handle, constraining controllers and pole vector constraints, the tool test handles even more features. In just this test alone, stretchy leg, vector calculated pole vector, and vector ball roll, which were all mentioned in previous sections, are applied to this leg for this test. The main purpose of the added features is to showcase the tool’s ability to accomplish tasks required of the manual setup while also adding on more advanced features, all faster than doing the basic setup by hand.

After both tests were administered and user times were recorded, participants were asked to fill out a Google Forms survey consisting of 11 questions regarding their rigging experience as well as the functionality and usability of the tool.

9.3 Results

The following section discusses the results obtained during user testing of the tool. The test results are divided into two categories: those dealing with the efficiency of the tool compared to manual rigging, and those dealing with the usability of the tool.

9.3.1 Time Efficiency

When performing user testing, as previously discussed, users were asked to perform a rigging test once manually and once utilizing the tool. Times were recorded for both of these tests. Users were also categorized into one of three subgroups based on rigging experience: professional for users whose job it is to rig, comfortable for users who rig occasionally (2-5 times a month), and beginner for those who had rarely or never rigged before. In Figure 30, regardless of the user's level of rigging experience, the tool performed between 2.03 and 6.34 times faster than users following the manual test. Additionally, even when testing the tool with a professional rigger, it performed 4.11 times faster than the user could manually.

User	Rigging Exp.	Manual Time (Min + Sec)	Tool Time (Min + Sec)	Speedup w/Tool
3	Professional	1:51	0:27	4.11
1	Comfortable	5:14	1:56	2.71
2	Comfortable	2:26	1:12	2.03
4	Comfortable	6:57	3:09	2.21
5	Beginner	10:15	1:37	6.34
6	Begginer	7:40	2:33	3.01

Figure 30. User speedup comparison between their manual setup time and their tool setup time. Sorted by level of rigging experience

On average, users performed the rigging test 3.40 times faster when using the tool. The average time for the test to be completed when using the tool was 1:49 minutes, while the average for manual rigging was 5:43 minutes. Using this data, a 90% confidence interval of [1.85, 4.95] of the average was calculated.

Faster Average	Tool Average	Manual Average	Standard Deviation	Variance
3.40	1:49	5:43	1.4772	2.619

Figure 31. Average, standard deviation and variance of tool rigging data.

This data suggests that the auto rigging tool will consistently outperform manual rigging, regardless of the user's level of rigging experience.

9.3.2 Usability

Following the rigging test, users were administered a usability form where they rated certain aspects of the tool, as well as Autodesk Maya's current rigging features.

The first of these questions concerned users rating Maya's native rigging toolset on a scale of 1 (poor) to 5 (ideal). The results were scattered, with 50% of users rating Maya's rigging tools at a 4, 16.7% of our users rating them as a 3, and 33.3% of our users rating them as a 2. This showed that there is not a consistent opinion among users about Maya's native rigging tools.

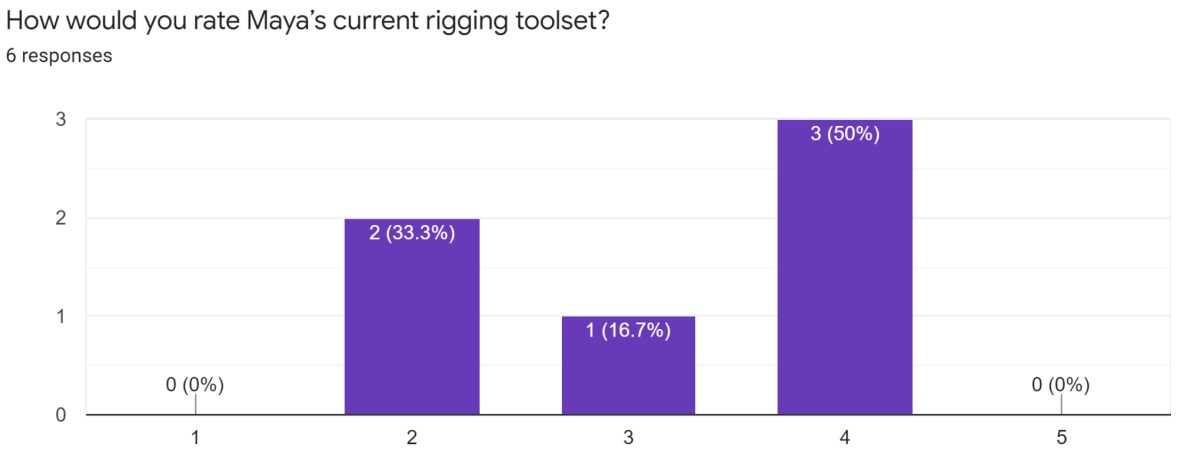


Figure 32. Users (y-axis) rating (x-axis) of Mayas current rigging toolset (1 = poor, 5= ideal)

The following question asked users to rate how the auto-rigging tool blends functionally into Maya, on a scale from 1 (poorly) to 5 (really well). In this case, the majority of users (66.7%) rated the tools functionality a 5, while the rest of the users (33.3%) rated it a 4.

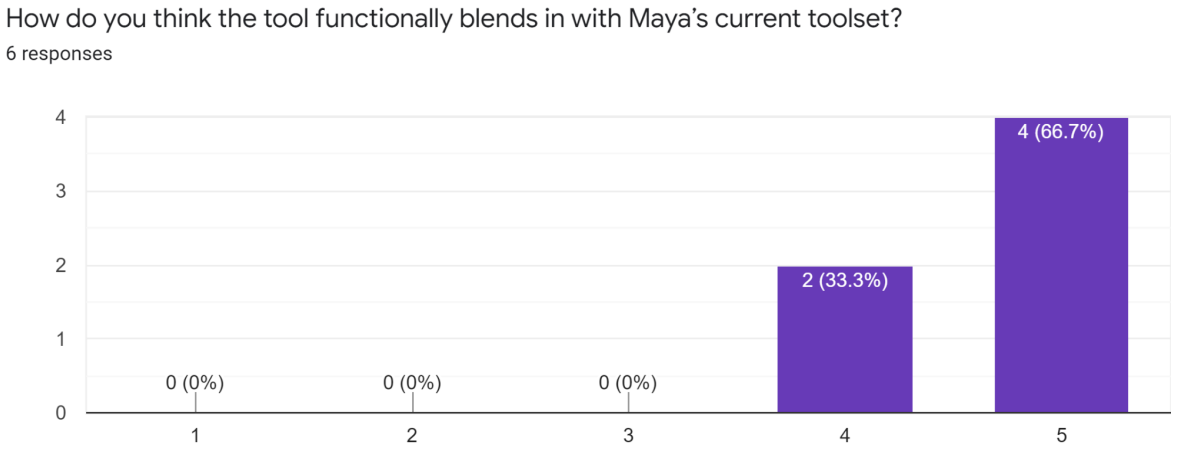


Figure 33. Users (y-axis) rating (x-axis) of the tool's functional blend into Maya (1 = poorly, 5 = really well)

Users were also asked to rate how the tool aesthetically blends into Maya, on a scale from 1 (poorly) to 5 (really well). In this case, 83.3% of our users rated the aesthetics of the tool as a 5, while the remaining 16.7% rated it as a 4. These results, as well as those of the above question, show how users believe the tool blends almost seamlessly into Maya's native user interface.

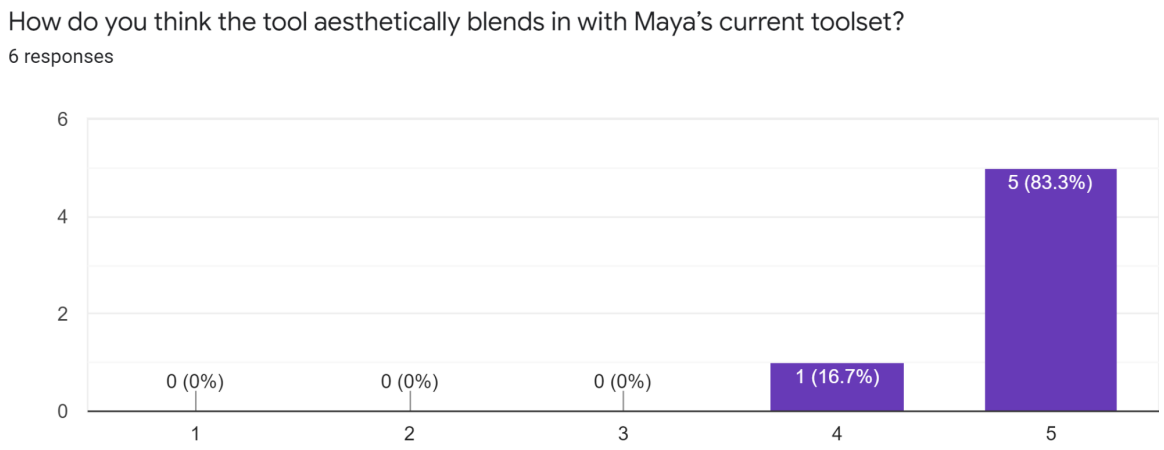


Figure 34. Users (y-axis) rating (x-axis) of the tool's aesthetic blend into Maya (1 = poorly, 5 = really well)

Finally, users were asked to rate the overall usability of the auto-rigging tool on a scale from 1 (poor) to 5 (great). 100% of our users deemed their experience with the tool great.

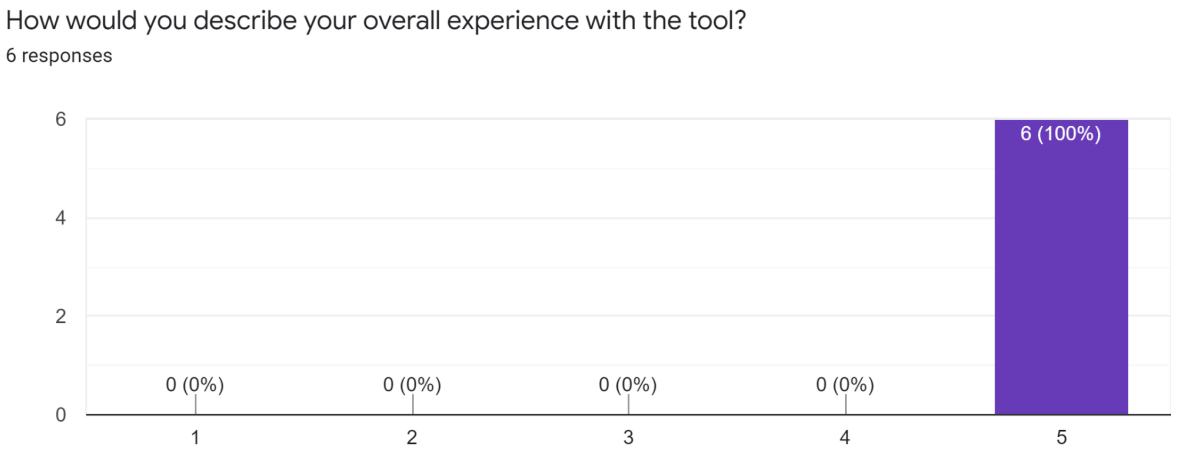


Figure 35. Users (y-axis) rating (x-axis) of their experience with the tool (1 = poor, 5 = great)

These results are promising, and show consistent satisfaction from the initial user pool with the usability of the auto-rigger. Overall, users consistently rated the tool positively in terms of both time efficiency and usability.

10. Future Work

10.1 Short Term Goals

In the short term, the tool requires clean up in order to be considered cohesive. There should be consolidation in naming conventions. In addition to this, the one feature that consistently works and is hooked up properly into the user interface is the leg script which handles features like basic IK setup, stretchy leg, vector calculated pole vector, and vector ball roll. The rest of the features like the face, spine, arm, as well as anything utilizing the ribbon system are close to being implemented into the UI as they can be applied with a simple function call.

After applying these features to the UI, additional testing needs to be taken care of. The leg side of the tool was the only part subject to user testing, which, while an essential part of rigging, was not indicative of all the features the tool can provide. With this in mind, different tests would need to be created to evaluate features like the ribbon and the face. These tasks are not beginner friendly, and it would be unrealistic to expect users to perform them by hand due to it creating a large time commitment compared to the 20 minute total test for the leg.

Furthermore, testing on whether or not our rigs are efficient for the animators is also important. All testing up until this point has been to see how convenient it is to set up for the rig artist, but it would also be valuable to know what animators think of the rigs generated by the tool. This will help fine tune additional User Inputs, future long term features that could be included at the request of testers, and whether or not an existing feature needs to be modified to make it more user friendly. Leveraging the input of those taking animation classes, particularly in 3D Animation I and II, would be helpful as the user base would at the very least be familiar with basic to intermediate Maya animation. At the moment those

classes utilize third party rigs that are found online, so it would be a good metric to see how our generated rigs compare to those.

10.2 Medium Term Goals

In regards to the face structure, the system needs systemic interaction. At the moment the Face Emoter and Eye Look system are both functional, but do not interact in any way. This is simply not the case when it comes to how the human eye actually functions. When people look up or down, their eye lids tend to follow.

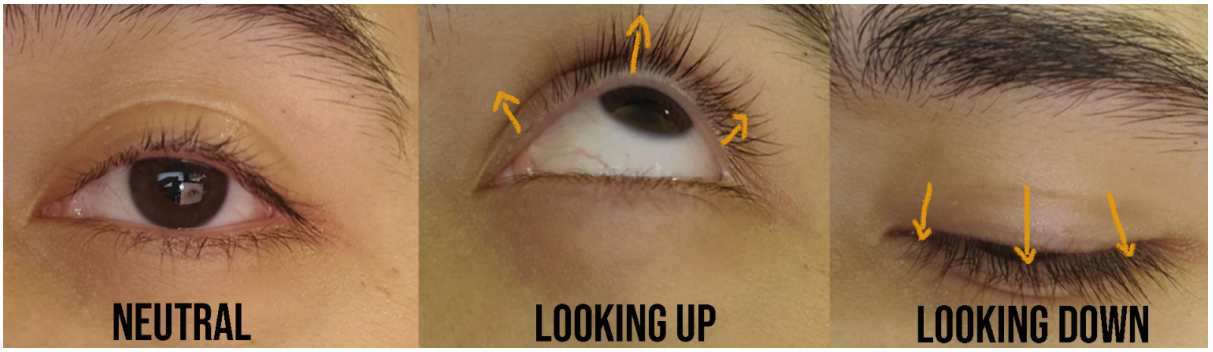


Figure 36. Eye look interaction with eyelid, real life example

The thought would be to have the eye look and face emoter systems interact, so that when the eye look controller aims up, the eyelids will automatically move up to as well as shown in Figure 36. The easiest way to approach connecting these two systems is through the main guide joint that both systems are parented to. The necessary controls needed to make this a reality are named after that base joint with a specific suffix, so it is very possible to find the needed components via string name search utilizing the name of the parent joint plus something similar to "_CTRL" at the end. Once assigned to a variable, this controller can now be utilized within the eye look system. The problem occurs where the user would need to be restricted from using the eye look before the eye face emoter for this specific method to work.

10.3 Long Term Goals

The long term goal would be to apply the tool to future IMGD MQP projects at WPI. In addition to this, the tool will continue to be used on sets of characters created by the Rigs of Color - Character Development team each year. MQPs and the Rigs of Color characters will be utilized to properly bug test the tool. This will inform the next tool development team on what features still need to be added and what edge cases still exist. The plan is to release the fully rigged characters resulting from Rigs of Color to the public so that there are more diverse and high quality rigs on the market. The tool will continue to be proprietary to WPI for the foreseeable future.

11. Conclusion

Rigging is an essential process in the field of 3D animation. However, manual rigging, often done in Autodesk Maya, is a tedious, time consuming and highly technical skill. Errors during hand rigging are common, even for advanced riggers, due to the repetitive and data heavy nature of the process.

To fix the problems that arise with hand-rigging, auto-rigging tools that automate or eliminate the rigging process have been created. Unfortunately, these tools often limit the user's creative freedom by providing a set of premade animations with little room for modifications.

In order to address the issues with hand-rigging and auto-rigging tools, we created a set of Python 3 scripts inside of Maya that provide most riggable elements in a 3D character, such as the face and body, with a graphical user interface to connect them. The scripts also allow users to rig any bipedal character with freedom to modify the rig guides.

User testing with participants of varying rigging levels showed that the tool, on average, outperforms manual rigging by being 3.40 times faster. Additionally, testing showed a high satisfaction rate among users, with 100% of participants rating their experience with the tool 5 out of 5.

12. References

Autodesk. “Matrix math in Maya.” *Autodesk Support,* 02 Feb 2022, https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2022/ENU/Maya-Basics/files/GUID-FE14C377-BD15-4BE3-8656-2A01CCF25D76-htm.html

“Adaptive Foot Roll Using Utility Nodes.” Adaptive Foot Roll Using Utility Nodes - Page 1 - Free Character Tutorials for Maya, *Highend3D*, https://www.highend3d.com/maya/tutorials/character/c/adaptive-foot-roll-using-utility-nodes.

Chery, Farley. “Motion Flow Rigging in Maya.” *Pluralsight*, 15 June 2011, https://www.pluralsight.com/courses/motion-flow-rigging-maya-73.

Giordano, Marco. “MAYA TUTORIAL: Cartoon eyeLid rigging” *YouTube*, 11 June 2015 https://www.youtube.com/playlist?list=PLqTYlr4mV7LeCo7q3uqfzO99oYSBidY3a

Hendrix, Greg. “Maya Applied Math - Placing Pole Vectors” *YouTube,* 13 Jan. 2018 https://youtu.be/bB_HL1tBVHY

Lesterbanks. “Check out This Unique Foot Roll That Always Has Contact.” *Lesterbanks*, 14 Feb. 2020, https://lesterbanks.com/2020/02/check-out-this-unique-foot-roll-that-always-has-contact/

Lesterbanks. “Understanding the Basics of Matrix Math in Maya.” *Lesterbanks*, 2 July 2017, https://lesterbanks.com/2017/07/understanding-basics-matrix-math-maya/.

Love, Jarred. “Custom maya matrix node constraint *UPDATED* rig tip” *YouTube*, 10 Dec. 2021

https://youtu.be/7A98wukXsJQ

Love, Jarred. “Transform negation *UPDATED* Maya tutorial” *YouTube*, 10 Dec. 2021

https://www.youtube.com/watch?v=HqXhNM0R8vo

Marcus, Sophia B, and Farley J Chery. “Farley Chery MQP Interview.” 27 Apr. 2022.

Mixamo, *Adobe*, https://www.mixamo.com/.

Petty, Josh. “What Is 3D Rigging For Animation and Character Design?” *Concept Art Empire*, 5 Oct. 2018, conceptartempire.com/what-is-rigging.

Zurbrigg, Chris. “PySide2 for Maya.” *Zurbrigg*, zurbrigg.teachable.com/p/pyside2-for-maya-vol-1. Accessed 26 Apr. 2022.

13. Appendices

Appendix A: Informed Consent Agreement

Informed Consent Agreement for Participation in a Research

Title of Research Study: Auto Rigging Tool

Introduction
You are being asked to participate in a research study. Before you agree, however, you must be fully informed about the purpose of the study, the procedures to be followed, and any benefits, risks or discomfort that you may experience as a result of your participation. This form presents information about the study so that you may make a fully informed decision regarding your participation.

Purpose of the study:

The purpose of this study is to compare the time it takes to rig a basic inverse-kinematic (IK) leg compared to the time it takes for our tool to accomplish a similar but more complex task.

Procedures to be followed:

You will be taking one of the following tests that should have been specified before viewing this document. PROCEDURE A, the rigging timed test and PROCEDURE B, the animation test and ease of use comparison.

See Procedures Below:

PROCEDURE A: Rigging Test

This procedure should take no longer than **30 minutes**.

In this study, you will be responsible for creating an inverse-kinematic (IK) based leg rig within Autodesk Maya 2022, both manually and then with our Tool. You will utilize a Maya scene with a premade Leg model already in the scene. You will be timed for both tests. We will be responsible for both the starting and ending of the timer. Once you are done rigging your leg in both tests please let us know when you are finished so we may record the time.

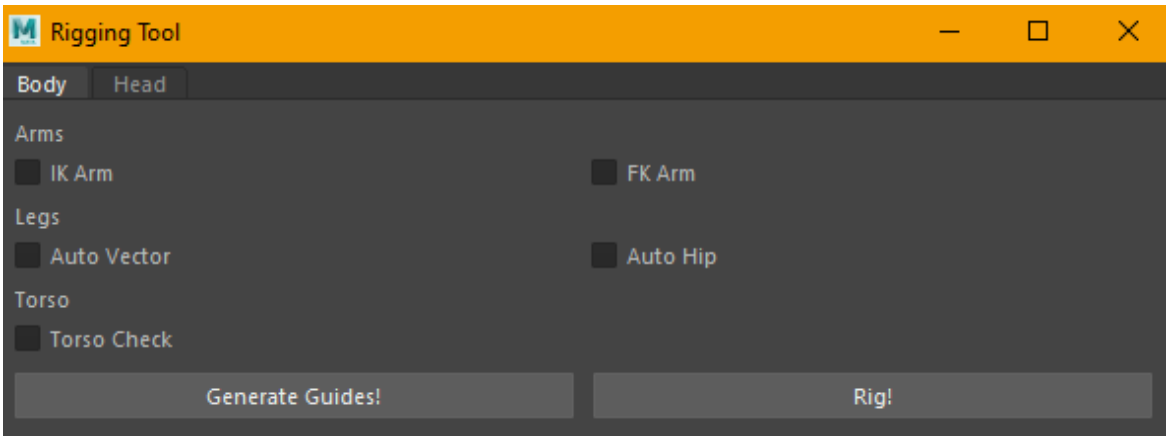
For the Manual test, you will:

1. Create a basic leg joint structure that matches the provided leg model which includes the following: thigh, knee, ankle, toe, toeNub. Please name the joints

in the Maya scene accordingly.
    - (Rigging Toolbar: Skeleton > Create Joints)
2. Create an IK Tool connection between the thigh joint and the ankle joint
    - (Rigging Toolbar: Skeleton > Create IK Handle)
3. Create a basic NURBS controller
    - (Create > NURBS Primitives > Circle)
4. Orient constrain the NURBS controller to the ankle joint
    - (Rigging Toolbar: Constrain > Orient)
    - Maintain offset must be on
5. Point constrain the NURBS controller to the IK handle generated in step 2
    - (Rigging Toolbar: Constrain > Point)
6. Create a locator and snap it to the knee joint by holding "V" and dragging it to the knee
    - (Create > Locator)
7. Move it out in front of the knee on one axis
8. Freeze the transformation
    - (Modify > Freeze Transformations)
9. Constrain the IK handle to the Locator using the Pole Vector constraint
    - (Rigging Toolbar: Constrain > Pole Vector)
10. Be sure to skin the leg with the now created joints
    - (Rigging Toolbar: Skin > Bind Skin)

For the Tool test, you will:

1. Open up our tool (we will inform you of its location before testing)
2. Press the **"Generate Guides!"** Button
3. Line up guides with the leg mesh provided by moving the leg gizmos in the scene
4. Under the "**Legs**" section, check the **"Auto Vector"** checkbox
5. Press the **"Rig!"** button
6. Be sure to skin the leg with the now generated joints **(Be sure to select the joints created in the Viewport and not the existing guides)**
    - (Rigging Toolbar: Skin > Bind Skin)



After both of these tests, we will provide you with a google form so you can provide feedback on your experience with our tool.

PROCEDURE B: Animation and Ease of Use Test

This procedure should take no longer than **1 hour**

For this test we will start by showing you a **short video** comparing the features between a **rig generated by our tool** and a **3rd party rig**. Both rigs will only feature the pelvis and legs. NO upper body animation will be displayed.

Once you are informed on all the needed features, you will be tasked with **creating a simple walk cycle** twice. Once with the 3rd party rig specified in the video and again with a rig that has been generated using our tools.

You will be provided with two Autodesk Maya 2022 Maya ASCII files to use as a base.

Animation Requirements:

- The animations must be 24 frames per second
- There must be Ball Roll included in both animation

If you have any questions on the requirements please let us know before beginning.

You will not be timed.

Risks to study participants:

There will be no physical risk to the participant during this research study.

Benefits to research participants and others:

This research study will provide Interactive Media & Game Development Playtesting Credit.

Record keeping and confidentiality:

We will be keeping two trial times. One of how long it takes for you to create an IK Leg manually without our tool and another of how long it takes to create an IK Leg with our tool. All of these times will be recorded on a Google Spreadsheet. To maintain confidentiality, the only information we will keep on record is your times specified above. These times will be shown to others and utilized in our data analysis, but they will not be attached to a name.

Records of your participation in this study will be held confidential so far as permitted by law. However, the study investigators, the sponsor or its designee, and, under certain circumstances, the Worcester Polytechnic Institute Institutional Review Board (WPI IRB) will be able to inspect and have access to confidential data that identify you by name. Any publication or presentation of the data will not identify you

Compensation or treatment in the event of injury:

This research study does not involve any actions that would bring harm to the user. Compensation and medical treatment will not be available in the case of an injury during testing. With this in mind, you do not give up any of your legal rights by signing this statement.

For more information about this research or about the rights of research participants, or in case of research-related injury, contact:

IRB Manager

Ruth McKeogh, Tel. 508 831- 6699, Email: irb@wpi.edu

Human Protection Administrator

Gabriel Johnson, Tel. 508-831-4989, Email: gjohnson@wpi.edu

Student Researchers

Sophia B Marcus, Email: sbmarcus@wpi.edu

Paloma González Gálvez, Email: pgonzalezgalvez@wpi.edu

Patrick Luck, Email: prluck@wpi.edu

Terry Deng, Email: tdeng2@wpi.edu

Your participation in this research is voluntary. Your refusal to participate will not result in any penalty to you or any loss of benefits to which you may otherwise be entitled. You may decide to stop participating in the research at any time without penalty or loss of other benefits. The project investigators retain the right to cancel or postpone the experimental procedures at any time they see fit.

By signing below, you acknowledge that you have been informed about and consent to be a participant in the study described above. Make sure that your questions are answered to your satisfaction before signing. You are entitled to retain a copy of this consent agreement.

_____ Date: _____ Study Participant Signature

_____

Study Participant Name (Please print)

_____ Date: _____ Signature of Person who explained this study

Appendix B: Usability Survey Questions

Below are the questions asked in the usability survey.

How would you rate Maya's current rigging toolset?

	1	2	3	4	5	
Poor	○	○	○	○	○	Ideal

What would you change about Maya's current rigging tools? Are there any tools you wish Maya had to aid you in the rigging process?

Long answer text

Have you ever used an auto-rigging tool or software (i.e. Mixamo)?

- ○ Yes
- ○ No

If so, describe your experience with the tool- pros, cons and features you would change.

Long answer text

What features of our tool did you find the most useful and why?

Long answer text

Did you have any trouble completing any of the tasks? If so, which one/s and why?

Long answer text

How do you think the tool functionally blends in with Maya's current toolset?

	1	2	3	4	5	
Poorly	○	○	○	○	○	Really well

How do you think the tool aesthetically blends in with Maya's current toolset?

	1	2	3	4	5	
Poorly	○	○	○	○	○	Really well

If you could change one thing about the tool, what would it be and why?

Long answer text

What additional features/changes would you expect to see in this tool in the future?

Long answer text

How would you describe your overall experience with the tool?

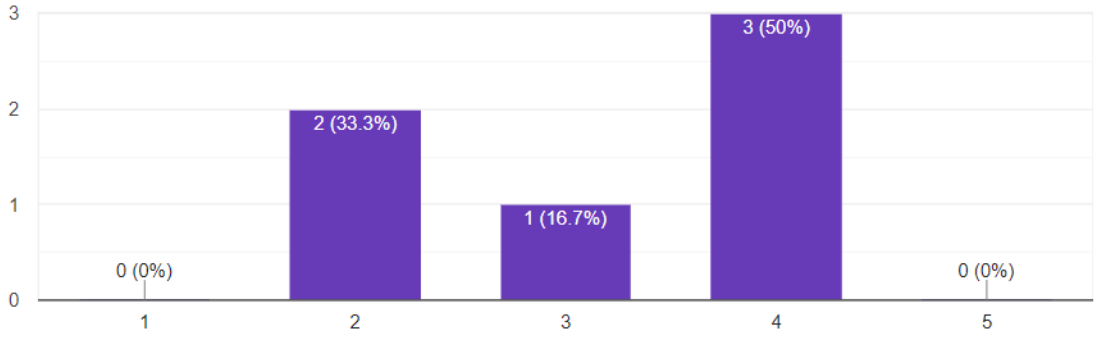
	1	2	3	4	5	
Poor	○	○	○	○	○	Great

Appendix C: Usability Survey Responses

Below are the responses to our usability survey.

How would you rate Maya's current rigging toolset?

6 responses



What would you change about Maya's current rigging tools? Are there any tools you wish Maya had to aid you in the rigging process?

5 responses

I would like for easier deformations.

Honestly I just think doing anything complex or anything that takes node editor is impossible for me so anything that does those complex parts automatically is really helpful. (ex. stretchy leg, I could not do that by myself)

better interface than the outliner to visualize rigged components, like a better node system for rigging

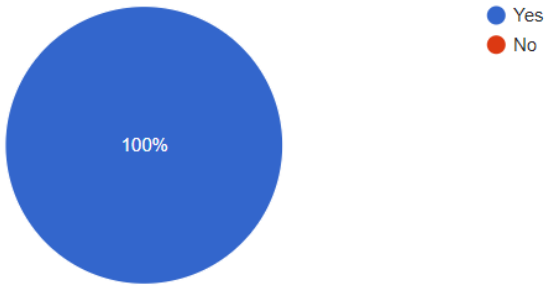
more support for pre-established rigging techniques, ie less reinventing of the wheel

I wish Maya had a mouth that could parse audio and lip sync it, that would be amazing

I don't know how to rig. I wish Maya had something to do quick rigs w/o having to learn the whole rigging process

Have you ever used an auto-rigging tool or software (i.e. Mixamo)?

6 responses



If so, describe your experience with the tool- pros, cons and features you would change.

6 responses

Mixamo was limited in it's skinning, but was quick and useful for getting animations onto a model.

I used Mixamo -- pro is that it is really easy to use and has pre-existing animations so you don't need to animate, con is that you can't really edit them easily (ig u still can but I don't know how to personally)

pros -works well with tweaking rig in maya, to correct skinning
-good at finding joints

cons -no controls included in rig, just bones and anim data
- no ability to customize bone setup in detail

I think its awesome, I with you could choose 5 joints like you do when creating them, that might improve the speed of having to move each guide. Other than that, amazing tool!

It was pretty cool to see a quick rig and animation. However as it was not customizablke and just had one rig with like predicted joints, sometimes the animation would not come out right or the limbs would bend weird

Pros- fast
Cons - usually messes up the fingers and you can't change it in mixamo

What features of our tool did you find the most useful and why?

6 responses

I liked how customizable the skeleton was.

SPEED, so much faster and also you don't have to remember all the complicated steps

added many additional helpful features in the rig that would have been time consuming to do by hand, ie. stretchy limbs, ball roll, etc

It had all the bells and whistles, which take so long to make.

I found the layout helpful, it was very clear what I was doing. Also, I found adjusting each joint helpful because you can better map it to a custom character etc.

being able to line up the bones was easy and intuitive. the ui was also simple and easy to understand

Did you have any trouble completing any of the tasks? If so, which one/s and why?

6 responses

I struggled understanding the leg controls after rigging at first.

I got a little confused because a whole human appeared when I was just trying to do a leg but it didn't really affect me completing any tasks

I wasn't sure exactly how to get started with the tool, but once I read the provided instructions again it made more sense

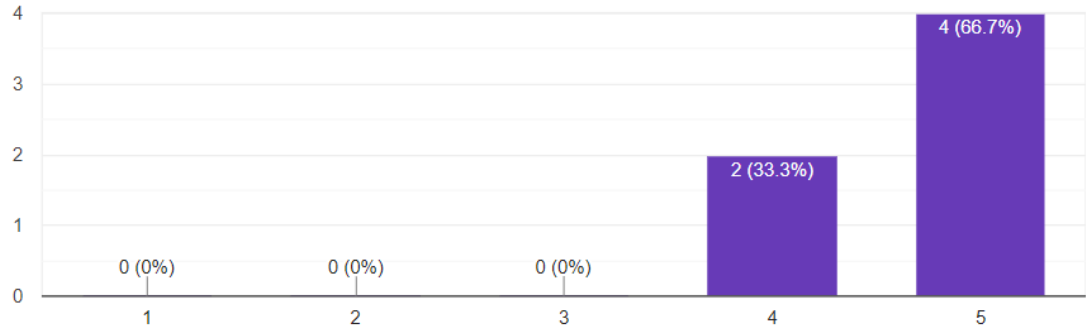
Finding each constraint, etc. in Maya

I didn't have much trouble with the tool it was very straightforward. I did have some problems with where stuff was and how to do stuff in the manual test bc I have not rigged before

no

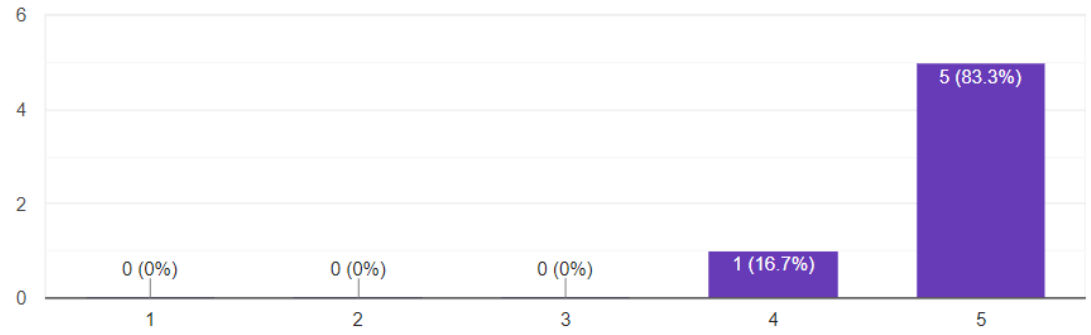
How do you think the tool functionally blends in with Maya's current toolset?

6 responses



How do you think the tool aesthetically blends in with Maya's current toolset?

6 responses



If you could change one thing about the tool, what would it be and why?

4 responses

Instructions on the tool for clarity of use!

i think more information could be presented on the main screen. maybe a label over the checkbox section that says "rig features", and more text to clarify what the guides are for ("create & adjust guides")

Clicking for joint creation rather than moving guides.

Idk what torso check means.. Also the tool icon is j the python snake. Also, maybe like resizeable window so I can snap into sidebar or smth

What additional features/changes would you expect to see in this tool in the future?

5 responses

Other body parts.

Being able to rig the whole body at once with this tool would be really nice. Although I'm assuming that is in the works

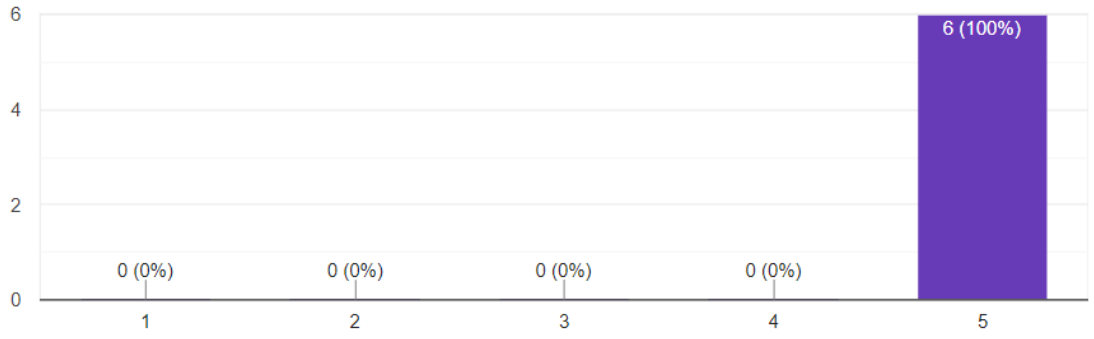
finger setup, since fingers have a lot of repetitive tasks associated with them

It's honestly amazing

Full body auto rig with adjustment

How would you describe your overall experience with the tool?

6 responses



Appendix D: Code

Use the following link to access the Python code for the project:

https://github.com/prluck/Maya_Rigging_Tool_2022