East Tennessee State University
Digital Commons @ East Tennessee State University

Undergraduate Honors Theses	Student Works

5-2018

Artist-Centered Technical Direction and Tool Development

Joshua Roberts

Follow this and additional works at: https://dc.etsu.edu/honors

Part of the Art and Design Commons

Recommended Citation

Roberts, Joshua, "Artist-Centered Technical Direction and Tool Development" (2018). *Undergraduate Honors Theses.* Paper 463. https://dc.etsu.edu/honors/463

This Honors Thesis - Open Access is brought to you for free and open access by the Student Works at Digital Commons @ East Tennessee State University. It has been accepted for inclusion in Undergraduate Honors Theses by an authorized administrator of Digital Commons @ East Tennessee State University. For more information, please contact digilib@etsu.edu.

Artist-Centered Technical Direction and Tool Development

By

Joshua Daniel Roberts

An Undergraduate Thesis Submitted in Partial Fulfillment
of the Requirements for the
Fine and Performing Honors Scholars Program
Honors College
East Tennessee State University

[illegible] 5-4-18
Joshua D. Roberts Date

[illegible] 5/4/18
Marty Fitzgerald, Thesis Mentor Date

[illegible] 5/4/18
Todd Emma, Reader Date

[illegible] 5/4/18
Dr. Scott Contreras-Koterbay, Reader Date

0 TABLE OF CONTENTS

1 INTRODUCTION

1.1 Purpose of This Document

This paper details the design and creation of two technical projects developed for Autodesk Maya to enhance the workflow processes of 3D animation. Both projects are aimed at solving problems typical to a technical director position and offer specific solutions for the purposes of character rigging and improvement of the animation pipeline, respectively. All details contained are intended to function 1.) as a technical design document, 2.) as a breakdown of development processes, and 3.) as a summary of results.

1.2 What is Technical Direction?

Technical direction is the process of solving problems within a specific field that require a high level of technical knowledge. In computer graphics and 3D animation, a technical director is typically an individual who develops technical solutions that are beyond the skillset of the team's artists. Their work may include character rigging, scripting to automate setup tasks, tool development, pipeline improvements, and general artist support. In senior-level roles, a technical director or technical artist is expected to understand the studio's entire process, from concept to delivery, on both an artistic and technical level, performing tasks required of both artists and programmers and acting as a point of contact between the two teams.

Technical directors serve an important function on a production team. To bring fantasy worlds to life with realistic settings, characters, and creatures, artists need access to effective tools. Sometimes the complex needs of a particular effect can be difficult to achieve out-of-the-box and require some research and development before they can be implemented correctly. Because artists tend to be somewhat lacking in technical skills, however, they are not able to produce these tools on their own. This is where the technical director comes in. The underlying concepts used to produce visuals for movies and games are dependent on mathematics, algorithms, and many other technical aspects. For everything they do, the artist is dependent on software which makes heavy use of these concepts, and technical directors have the know-how to build within these constraints to create something new that enables the artist to better do what they are trained to do.

1.3 Summary of Projects

The two projects broken down and presented as part of this paper are designed and developed to serve the animation process. The first project, a Hybrid FKIK Tentacle system, is an exercise in character rigging. For this project, a new method is derived of interacting with and controlling joints to achieve motion beyond that of a typical IK system. The second project is a fully-integrated plugin developed for Autodesk Maya using python 2.7 and the PySide2 UI Framework. The tool was developed to solve the control problem that the animator experiences when parent-constraining grabbable objects in Maya.

The development of each project follows a similar evolution, and the documentation of each is broken down into the following sections: A background on the project, an executive summary of project goals and design considerations, a breakdown of the development process from beginning to end, and a summary of results along with a critical analysis of the project's function as concerned with the delivery of its goal. Each project was approached with the prior understanding that its function would fulfil a specific need and thus would require a certain sensitivity to the contextual application of its results, including specific parameters outlined in the design of each project.

1.4 Statement of Goals

While the primary purpose of the projects reported in this document is concerned with the processes of 3D animation, their development aims to highlight the underlying design and conceptual work required for projects in similar technical fields. Throughout development, all results were designed with human use in mind, giving special attention to maintainability, simplistic design, support for future applications, and generalized problem-solving.

Both projects have been developed fully from concept to creation as a part of this Thesis and involve tools and techniques both familiar and unfamiliar at the time of creation, chosen specifically because they provided a difficult problem to solve. From the beginning, the intention was that these would force me to learn new technologies efficiently and effectively in order to understand best practices and their applications toward solving the problems detailed below.

2 HYBRID FKIK TENTACLE RIG

2.1 Background

During the Fall semester, I was approached with a character rigging problem by students enrolled in the game production class. The character, named the Creeping Chaos, possessed six tentacles at its base and humanoid features for the upper body (as seen below in Figure 2.1). The character was designed as an enemy in their game, inspired by the stories of H.P. Lovecraft. They wanted the tentacles to function in a specific manner, seemingly propelling the character forward while also allowing for versatility and control in the animation process. This motion would require movements such as a Wave, a Curl, and a Displace. I was tasked with discovering an appropriate method to achieve this type of motion.

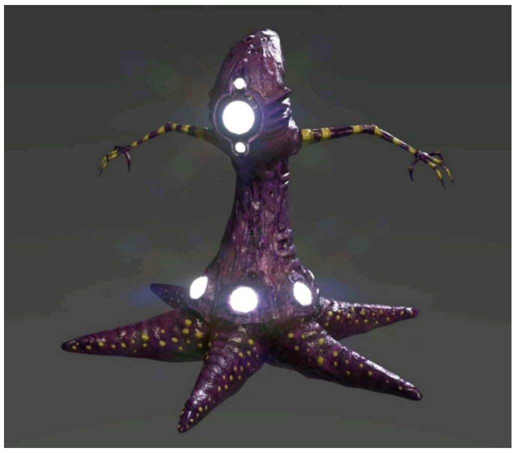


Figure 2.1: The Creeping Chaos

2.2 Project Summary

After accepting the challenge posed by the Creeping Chaos, I decided to summarize my goal into a purpose statement based on what I knew:

> *I will design, prototype, and integrate animation controls for each of the six tentacles of the Creeping Chaos into a fully-functional character rig, allowing the tentacles to wave, curl, and displace.*

Although, with only this statement to go by, the problem is still a bit vague. What exactly is meant by fully-functional? What type of motion will be produced by wave, curl, and displace? How will the animator control this motion? These things must be defined before starting development. Curl is conceptually easy to grasp, and with some prior knowledge I know that I can achieve this type of motion with a basic forward-kinematics system. I also know that basic motion of a wave can be achieved using the sine function as a nonlinear deformer, having a specific amplitude and offset. When approaching displace, the most complicated of the three behaviors, I spoke with the game team to clarify what they wanted the rig to accomplish and tried to think about the context in which the animator would be using the Creeping Chaos. I then settled on interpreting displace as the offset observable in the tentacle at a specific point when laid over an object on a flat surface.

2.3 Design

With a more clearly defined purpose statement and a basic understanding of how the rig should work, I was able to break the problem into smaller pieces. To build a reliable and fitting solution to the initial problem I outlined the following parameters.

Intended Audience

An animator who possesses minimal technical knowledge and will need to infer all relevant information based on visual elements.

Desired Behavior

Behavior	Definition	Achieved By
Wave	A sine wave with specific amplitude and offset	Nonlinear Sine
Curl	A simple bend from root to end	FK System
Displace	Offset of the tentacle at a specific point when laid over an object on a flat surface.	Unknown (Expected: some type of IK Spline System)
Fully-Functional	Possessing both FK and IK behaviors, with controls which can be manipulated without deforming in an unrealistic way, and allow for switching between each of the contained systems.	Hierarchy and Build

Working Environment

For its initial function as an animation tool, the tentacle will be used in the 3D viewport of Autodesk Maya, while the end result (the animation) will be exported for used inside of Unreal Engine 4.

Goals

The final rig must demonstrate simplistic use to the animator, which will be defined as necessitating a minimal adjustment of settings on the animator’s behalf to fully operate any component.

Considerations

Because the resulting animations will be exported to Unreal Engine 4, the tentacle must be built to contain a single-joint hierarchy of bind bones. This allows animations to be baked down for easy export. To maximize the return on investment, the solution will have to be viable for future use cases. Additionaly, the Creeping Chaos has six tentacles, so the rig will need to be modular as well, that is, self-contained and able to scale and transform while retaining functionality for each tentacle. This way it only has to be built a single time and can be imported for each consecutive instance.

2.4 Process and Development

Throughout the development of the Hybrid FKIK Tentacle, there were two main sources of difficulty: 1.) Figuring out how to control joint displacement (conceptually) and 2.) Figuring out how to build the whole rig and organize the input of each of the 3 systems: Curl, Wave, and Displace, while maintaining flexibility and simplicity for animation.

Solving the Displacement Problem

To control joint displacement from an animation perspective, I decided to place three controls along the length of the tentacle which could be offset perpendicular to the tentacle's forward axis. To accomplish this, I set up two curves inside of Maya, one to act as an IK Spline and another to act as a blend shape for the first. The second curve was given skin clusters corresponding with each joint on the tentacle to allow for maximum flexibility.

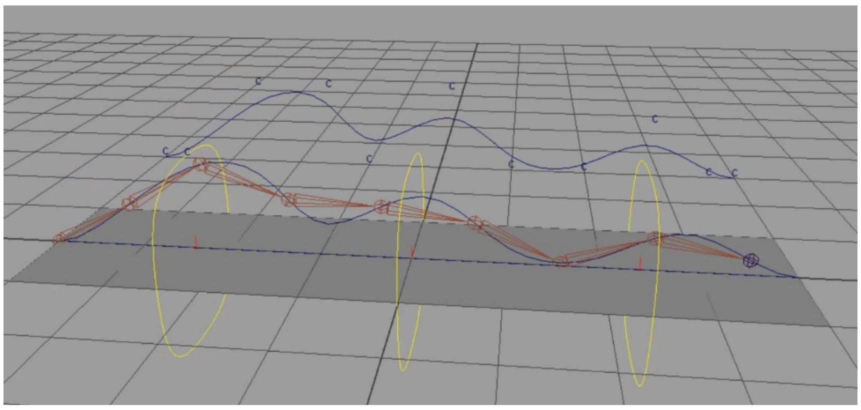


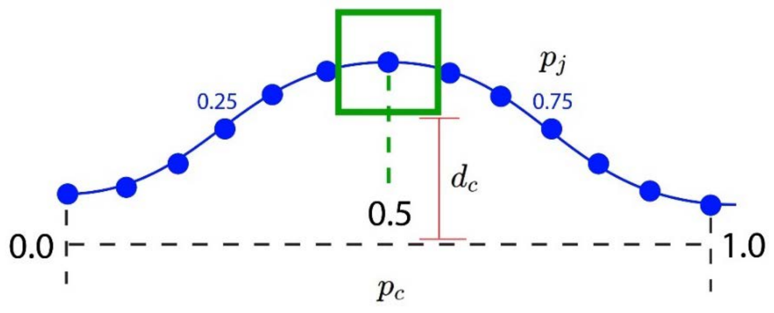
Figure 2.4a: Cluster Deformations

One major problem with this approach, however, is that each cluster must be moved individually. To maintain full control of the rig, the displacement would have to have as many controls as clusters. This is far less than optimal, especially considering that the Creeping Chaos is limited in resolution at 11 tentacle joints, whereas future uses may be much higher resolution and possess 20 joints or more.

It would be beneficial for the goal of simplistic design if the rig could have a specific number of controls which drove an arbitrary number of joints. Because each control would have to drive multiple joints, this would mean that the displacement would have to be proximity based. This would allow the controls to slide up and down the tentacle freely and have a determined falloff that could be adjusted by the animator in real time. Given the goals and purpose of the project, this seems to be a viable solution to the displacement problem.

To allow for control sliding, I took inspiration from the design of Jeff Brodsky's variable FK Trunk Rig [1]. The initial spline curve was extended to create a surface matching its resolution and providing a normal to the tentacle's curvature. For the purposes of the Creeping Chaos, three displacement controls were created along the length of the tentacle. A single hair follicle was placed on the surface for each control as they were grouped. The resulting group was parented to its respective follicle and translation along the length of the tentacle (perpendicular to the normal) was locked to allow for a calculable position based on a custom attribute which drives the position of the follicle along the newly created surface.

The use of follicles brings advantages both functionally (surface sliding) and mathematically. The surface position of each follicle is measured from 0 to 1 with 0 being at the base of the tentacle and 1 being at the tip. Because of this, the relative position of each joint can be calculated simply by dividing 1 by the total number of joints – 1 (to account for joint 0). These relative positions can then be stored as custom attributes on every joint and then be used to drive other values. Now, with a means of determining relative joint position, control offset, and control displacement, I was able to formulate an equation for calculating the falloff.



To find the relative joint displacement $\triangle x$

$$\triangle x = (1 - \frac{|p_c - p_j|}{f}) * d_c$$

$$|p_c - p_j| = dist_j \begin{cases} dist_j < 0 & dist_j = 0 \\ dist_j > f & dist_j = f \end{cases}$$

Figure 2.4b: The Displacement Equation

To find the displacement of each joint, all that must be found is the relative distance from the control to that joint (to which an upper and lower bound is imposed), divided by the falloff distance. This provides the proportional distance of the joint within the falloff range. Inverting the proportion, we now have the appropriate influence amount and need only multiply this by the displacement of the control to achieve the final displacement result. In this example, the falloff is linear, but when implemented within Maya using nodes, the upper and lower bound is calculated by a Remap Node, which provides the option to use a curve-based interpolation.

Solving the Build Problem

Now that the displacement controls are working properly, the remaining components must be arranged in a way which allows for the best functionality. Because we will be using a nonlinear sine deformer for the wave effect, this can be directly applied to the IK spline used to drive the deformation joints. Adding two deformers, one for the up axis and one for the secondary axis, provides for the most flexibility of motion in the rig.

With the current build, the sine controls drive the joints, but the displacement controls do as well. Both effects add together, which is okay from a functionality standpoint, but the controls do not travel with the wave. This can be solved by also applying the nonlinear deformers to the surface created in the previous problem to hold follicles and drive the position of the displacement controls. Now the controls will travel with the wave and give an accurate visual representation of how they are functioning for the artist.

Lastly, the curl controls must be created. Because the currently available joints are using an IK spline and we know the curl will use FK, a secondary joint chain must be created with simple FK controls. With consideration to my initial goal of simplicity, I decided to by default show only a small number of FK controls which make larger curling motions and provide the option for displaying a control for every individual joint should the animator want more control over the rig. Because the wave is already set up to drive the displacement system, the curl's FK system can be applied in the same way. The IK spline and the displacement surface can be skinned to the FK joints, and then the same offset is achieved. To prevent undesirable results, the wave input must be applied to deformation first. The final construction of the tentacle's systems is laid out in the diagram below.

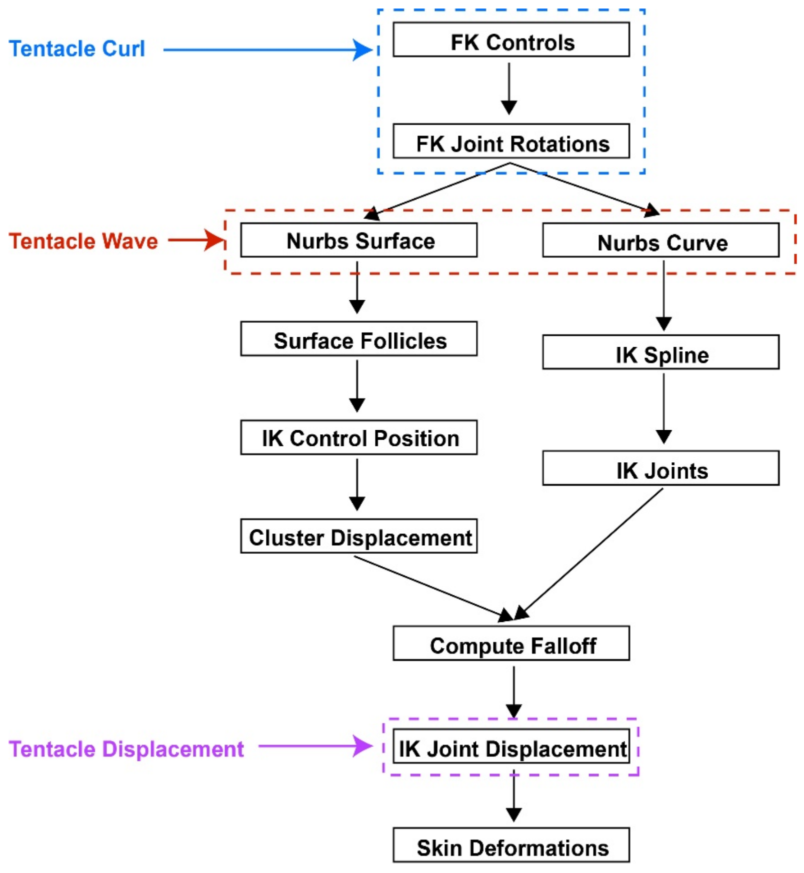


Figure 2.4c: Tentacle Build

Not pictured: the nonlinear sine deformers apply to both the Nurbs Curve and Surface, before the FK Joint Rotations in the deformation order. The diagram is shown from the perspective of user-input.

2.5 Results

The build that I was able to achieve for this rig resulted in a Hybrid FKIK setup, meaning that the tentacle is in both FK and IK control modes at the same time, requiring no mode switching for the animator to reach the desired controls. This result effectively meets the goal of simplicity from the user side, and a clean construction allows components to be organized in a way that supports full scale and translation of the rig while maintaining functionality, which accomplishes the goal of modularity.

Limitations

The current build uses Maya's remap node to create a linear falloff. To change this, the output curve of the remap node would have to be changed for every single joint as it is connected to every single control. For this to be clean, the curve on every single remap node would have to be the same shape. Since the rig was partially generated using scripts, this process done by hand would be very tedious and inaccurate. If the falloff curvature is to be changed after the rig is built, a better solution must be found.

Visual limitations of the tentacle rig are specific to its usage in the Creeping Chaos. Designed for a game, the mesh is rather low resolution, providing for only 11 joints in the current rig. This number of joints is not adequate to smoothly deform the tentacle when sliding displacement controls. A higher-fidelity tentacle would require at least double the number of joints as are present in the Creeping Chaos rig. This would not prove to be a major issue though, since the displacement equation is based on a position ranging from 0 to 1. The displacement will still be valid no matter how many joints are present. All that must be done to achieve a higher resolution is to abstract the generation scripts to support an arbitrary number of joints.

3 MAYA PARENT CONSTRAINT MANAGER

3.1 Background

A common problem in character animation occurs when a held object must be passed between a character's hands. Any time this must occur, the passed object (or child object) must be constrained to both hands and switch between them at points in time defined by the animator. After speaking with a few animators, I knew that a solution to this problem would be useful for several applications and add value to the animation pipeline.

Within Maya, a method of constraining an object to multiple parents already exists. However, it is less than optimal. The animator must first position the child object where they want it to be when it is weighted to the first parent and then add the constraint. This must be repeated for every desired parent that will interact with the child object. The result given by this method is very rigid, as the relative position of the child when the constraint is active can no longer be changed after setup. Additionally, setting the constraint on the child object itself causes the constraint's output to feed directly into the child's translation channels, competing with the animator for control. If the animator would like to animate the child object freely, they must first set a key on the grabbable object to prevent visible popping, deactivate all parent influences, and then animate the object. To switch between parents, the animator must set keys on all weight channels and the child object, then set the weight of the current parent to zero. The animator must then move the child object towards the next parent to make the transition as smooth as possible (all manually) and then set the weight of the next parent to 1. There is a lot of back and forth with adjustment of settings when using this method, which can very easily distract the animator from their work. I wanted to create a better process.

3.2 Project Summary

I wanted to create a tool which managed the parent constraint process for the animator. I laid out the purpose of this project as follows:

> *Design and implement a tool, which is integrated within the animator's workspace in Maya, for the purposes of managing parent constraints and relieving the animators of any burden caused by these processes.*

The first part of this purpose statement is straightforward, and the second gives some clear direction, but the specifics of the tool still must be fleshed out. For example, which function of the parent constraint do I want to manage and what exactly do I expect a burden on the animators to look like? While burdens can simply be defined as 'unwelcome processes', I think that it is better to define them in context of the whole statement. Specifically pertaining to this project, I expect repetitive setup tasks to be minimized and workflow processes which do not properly integrate with the overall character animation workflow to be replaced with something more intuitive in the given context.

3.3 Design

With a clear direction for the project and a basic underlying goal I was able begin layout out the design elements. Specifically, I wanted to retain clean animation curves on the child object and remove as much value-changing as possible from the animator's workflow (such as when switching weight values).

Intended Audience

An animator with an artistic background and minimal technical skills. They must be able to intuit the functionality of the tool visually with no additional guidance.

Desired Functions

- Manage all constraint setup for the animator in as few steps as possible.
- Automate the parent-switching process with a single-click.
- Maintain clean animation curves, meaning that the tools should affect those curves as little as possible.

Working Environment

Within the Maya viewport, specifically for use with the animation workflow. Pictured below in Figure 3.3, all tools necessary for animation are highlighted.

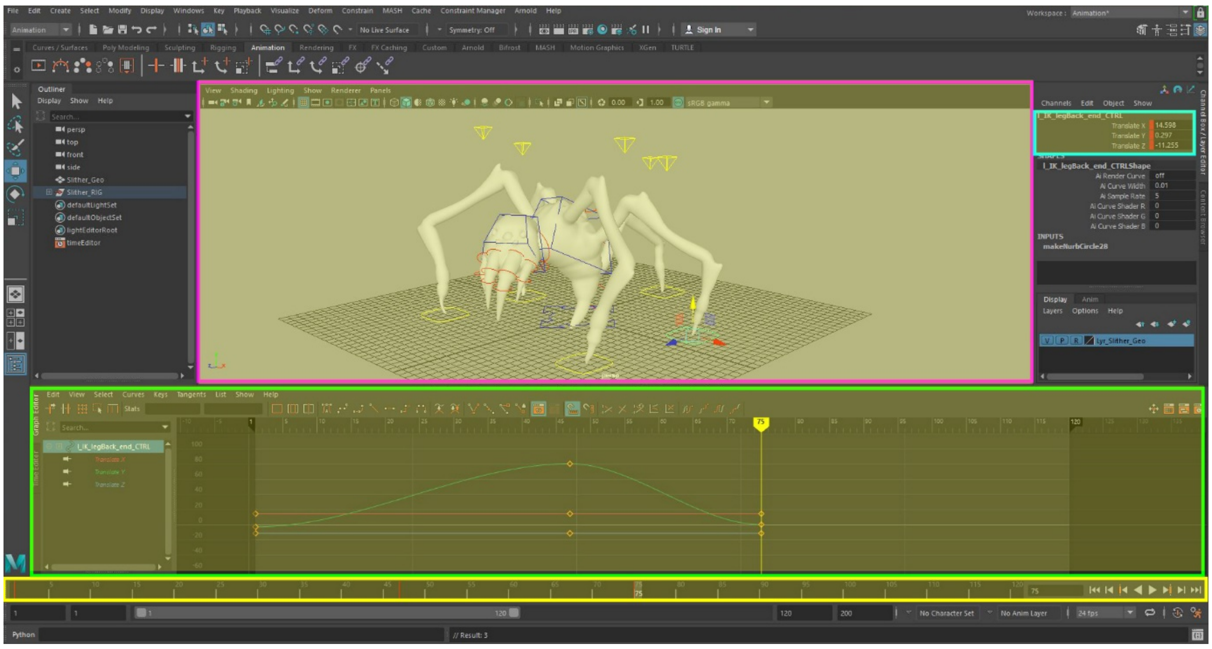


Figure 3.3: The Animation Workspace

Goals

- Minimize repetitive setup and automate as much as possible.
- Aim for simplicity; don't overcrowd the workspace with information unless relevant to the current visual context and use language that the user (the animator) uses when talking about their work.
- If passed on to a new user, they should be able to understand how to use the tool in 5 minutes

Technical Requirements.

The constraint manager tool will be developed as a Maya plugin entirely using Python 2.7. This will employ my previous knowledge of Maya Python and PyMel. Also for use with this project, I will have to learn PySide2, which is a library containing bindings to the Qt5 UI Framework and is included with Maya.

Considerations.

Because the tool will be integrated with Maya's UI, its design will have to consider the existing tools used in the animation workflow. Any new UI designs created as for the constraint manager will have to match with Maya's existing user-interface conventions. Knowing this, how can I balance the tool's functionality with its accessibility? In my experience with the development of previous tools, sometimes the simplest solution to a problem is not the one that makes the most sense for its audience. For example, when provided with a quick select button which selects all keyable objects in the scene, animators were confused as to its usage. They did not expect that button to exist, as group selection is something they must do on a regular basis. Because of this, the animators felt more comfortable selecting each item the way they select other item groups, and as a result the button could be removed from the tool to simplify the GUI.

Knowing that the constraint manager will be developed for an audience with an artistic skillset rather than a technical skillset, I must consider that most artists will usually try only once to understand a new tool. If they can't easily internalize the concept of how something works, they rarely dig deeper than surface level and the automatic response is to assume that it is too technical for them to understand. After an experience like this they are more likely to revert to their old existing workflow than give the new way a second chance. To circumvent this effect, the new way must be easier to understand than the old way.

3.4 Process and Development

To approach the initial problem of constraints, the tool nests the child object within two groups (Figure 3.4a). The outer group is directly constrained to the parent objects while the inner group is used to calculate the offset of the child. Using this method, switching parents is a simple as recording the child's world position, swapping the parent weights, and then moving the offset group to the child's previous position that was recorded. This has three big advantages. First, the child object's transformation channels are kept clean, since they are not modified directly. The child also maintains the same relative position within the inner group, so no drastic changes are seen in the 3D viewport. Second, the offset group can be repositioned freely. This solves the problem of rigidity and allows the relative parent position of the child to be modified easily without affecting any other process. Lastly, since the offset is calculated every time, the child object is able to remain in the same world space position from frame to frame while switching between active parents. This means unpleasant blending or popping due to parent positions is completely eliminated.

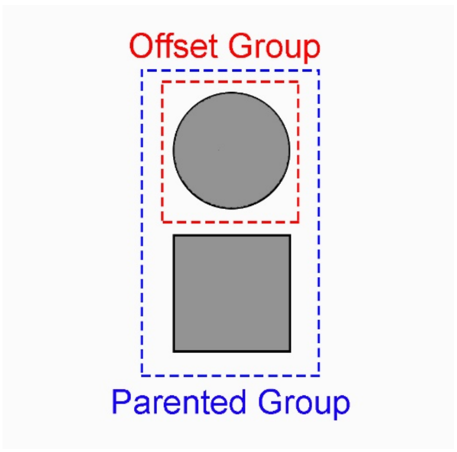


Figure 3.4a: The Child Object (Sphere) with nested groups

User-Interface Design

Based on the animation environment laid out in the design section above, the animator's focus is primarily centered on the lower-middle portion of the workspace. As a result, I decided that most accessible and appropriate location for the tool would be docked below the 3D viewport, right in the middle of the screen, above the time slider and the graph editor (as in the animation workspace view). As such I outlined the functions required of the user-interface and began laying out the design of my tool.

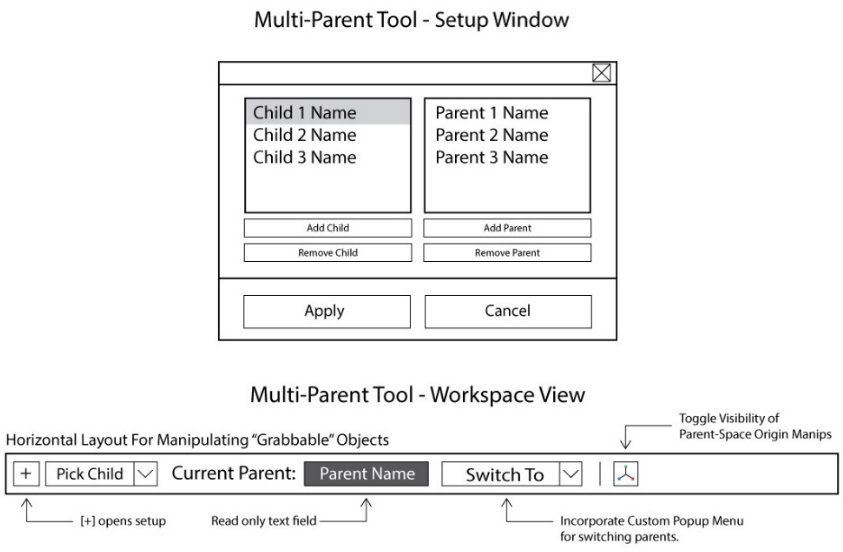

Multi-Parent Tool - Setup Window
Child 1 Name
Child 2 Name
Child 3 Name
Parent 1 Name
Parent 2 Name
Parent 3 Name
Add Child
Remove Child
Add Parent
Remove Parent
Apply
Cancel
Multi-Parent Tool - Workspace View
Horizontal Layout For Manipulating "Grabbable" Objects
Toggle Visibility of Parent-Space Origin Manips
+
Pick Child
Current Parent:
Parent Name
Switch To
[+] opens setup
Read only text field
Incorporate Custom Popup Menu for switching parents.

Figure 3.4b: User-Interface Wireframes

The design of the setup window pictured was chosen to satisfy the integration goals of the tool, since Maya contains several similar windows which look and function identically to this design. The goal is to minimize the amount of learning that is required of the animator when they try to the constraint manager tool.

Plugin Architecture

The organization of the various components of the tool is very important for its continued development. The architecture detailed below employs software design best practices to maximize the maintainability and readability of the code elements.

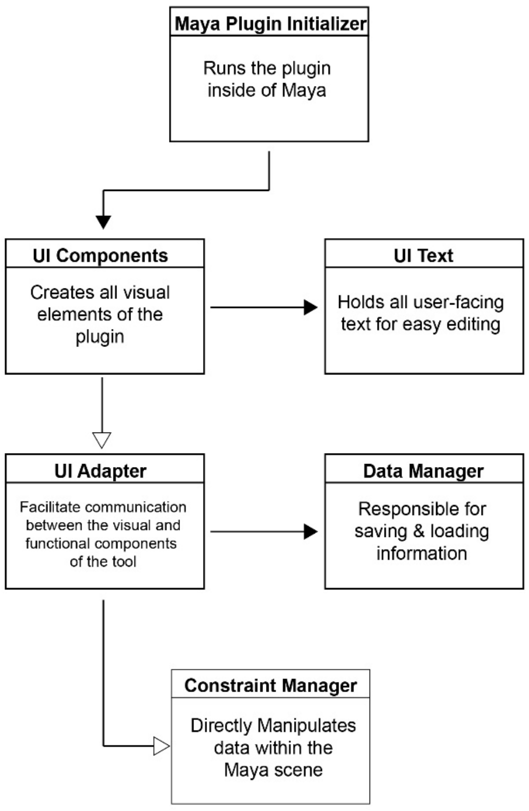

Maya Plugin Initializer
Runs the plugin inside of Maya
UI Components
Creates all visual elements of the plugin
UI Text
Holds all user-facing text for easy editing
UI Adapter
Facilitate communication between the visual and functional components of the tool
Data Manager
Responsible for saving & loading information
Constraint Manager
Directly Manipulates data within the Maya scene

Figure 3.4c: Constraint Manager Plugin Architecture

In the design, everything is encapsulated. This means that all respective components of the tool are organized into containers (files) which contain all code that relates to that process. For example, all of the UI Code is housed in ‘UI Components’, while all of the functional code which directly manipulates object in the Maya scene is held in ‘Constraint Manager’. If I want to update the label on a specific button, all I have to do is change a value in ‘UI Text’. This way, when debugging or modifying the plugin, it is much easier to find certain sections rather than navigate 1200+ lines of code.

Designing for Simplicity

In the initial state of the tool, only a single button is displayed: the setup option. After that button is pressed and the setup is completed, the menu changes contextually when the new child object is selected. This makes it obvious that the displayed information is associated with the currently selected object, and the animator only has to understand one input at a time.

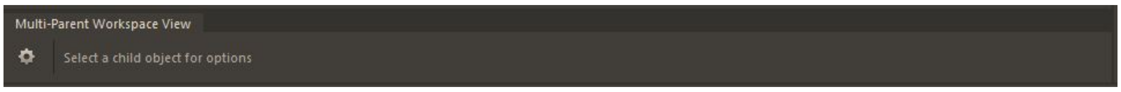

Multi-Parent Workspace View
Select a child object for options

Figure 3.4d: The Initial Visual State

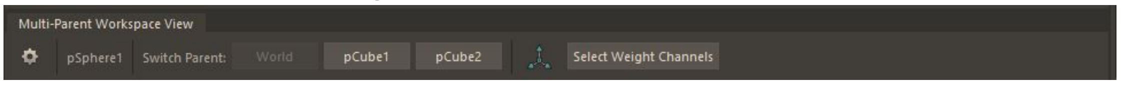


Figure 3.4e: Contextual Menu Items

3.5 Results

The final result of the constraint manager allows the animator to complete the entire setup process in one step, switch parents in one step, and displays information contextually to prevent against cluttering the workspace. All of these factors combined allow the plugin to achieve its goal of simplicity and accessibility.

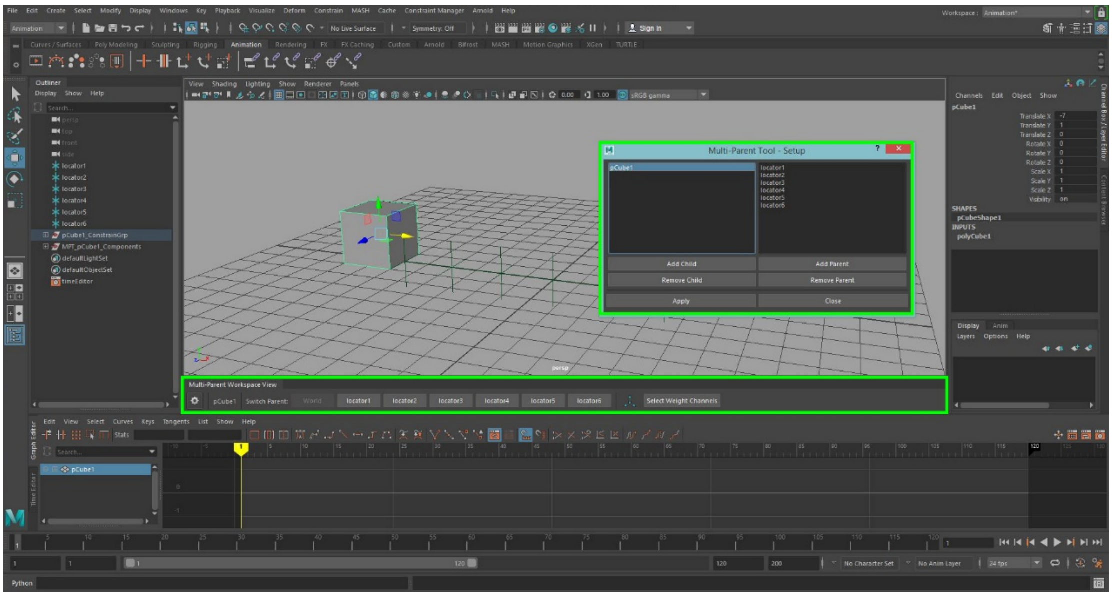


Figure 3.5: The Finished Tool in Maya

Limitations

Because the tool groups the child object before performing parent-switching operations, modifying the weights directly is not the most straight-forward of tasks. Selecting the parented group specifically is required to modify these animation curves and as a result, adjustments are simple but the process is obscured. In future versions, a means of exposing this functionality to the user will be necessary to ensure the continuing benefit of this tool in the 3D animation process.

4 REFERENCES

[1] Brodsky, Jeff, (2013). *Jeff Brodsky's trunk rig demo*. Retrieved from https://vimeo.com/49353110

[2] Cole, Mike. (2015, June 5). *Advanced PyQt for Maya*. Retrieved from https://www.pluralsight.com/courses/advanced-pyqt-maya-2122

[3] Gamma, Helm, Johnson, & Vlissides. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley. Pp. 139ff. ISBN 0-201-63361-2

[4] Next Day Video. (2013, March 20). *Transforming Code into Beautiful, Idiomatic Python*. Retrieved from https://youtu.be/OSGv2VnC0go

[5] PyCon 2015. (2015, April 11). *Raymond Hettinger – Beyond PEP 8 – Best practices for beautiful intelligible code – PyCon 2015*. Retrieved from https://youtu.be/wf-BqAjZb8M