RE-EVALUATING VFX WORKFLOWS:
ANIMATING VEHICLE DYNAMICS

A SENIOR PROJECT

PRESENTED TO:

THE FACULTY OF LIBERAL ARTS AND ENGINEERING STUDIES

CALIFORNIA POLYTECHNIC STATE UNIVERSITY, SAN LUIS OBISPO

IN PARTIAL FULFILLMENT

OF THE REQUIREMENTS FOR THE DEGREE

BACHELOR OF ARTS

BY

CHARLES DUONG

DECEMBER 2014

TABLE OF CONTENTS

LIST OF FIGURES

Introduction

Brief History of Film & Visual Effects

The idea of creating moving images can be traced back to the first half of the 19th century. In 1829, the Phenakistiscope, an invention that consisted of a disk with sequential images lined along the outer rim, created the illusion of movement when rotated. Four years later, the modern zoetrope built upon this idea and shifted the images onto a vertical strip that wrapped around the edge of the disk. The images lined up the interior of the zoetrope and vertical slits were placed above the images to provide a view of the animation within.

The first time something classified as a "real" motion picture was displayed for public audiences was when the Lumière Brothers showcased '*La Sortie de l'Usine Lumière à Lyon* (*Workers Leaving The Lumière Factory in Lyon)'* in 1892. That same year, Alfred Clark directed possibly the first special effects shot, *The Execution of Mary, Queen of Scots*. Visible within the scene is what is now known as a 'cut edit' where the frame cuts out, the actor is replaced by a mannequin, and then the shot continues from this point in the next immediate frame. Several years later, the pioneer of visual effects (VFX), Georges Méliès, took these concepts, and was inspired to create a studio that produced more than 200 short films, all using innovative compositing tricks. From cut edits, to matte paintings, to color correction, he developed new ways of looking at moving pictures. These leaders laid out the foundations for today's film and VFX industry, and more than a century later, most of the digital techniques used today still derive from their work. (fxPHD, 2013)

Current State of the VFX Industry

With the amount of technical innovation that has occurred within the past several decades, the VFX industry has gone through a multitude of change in this digital era. One would think the industry is booming, but in the past decade, one VFX statistic stands out above the rest, and it is not a cheerful one:

> "Between 2003 and 2013, 21 visual effects companies closed or filed for bankruptcy, yet of the 50 highest grossing films of all time, 49 are what one might call 'effects films'" (Seymour, 2014).

This statistic is due to number of reasons; however for this analysis, I will be focusing on the VFX budgeting, bidding, and overall production business formula established decades ago. Developing an accurate budget proposal for a VFX shot is a daunting task, due to unforeseen variables that will eventually come into play. When it comes to creating "something that no one has seen before," it is difficult to gauge the final pricing for a product that was merely a concept during the bidding process. (Seymour, 2014)

It has been common practice for the VFX house to establish a "fixed bid," a set price for their final product, rather than an hourly rate for their work. Realistically, this bid is eventually adjusted as approved overages when the VFX studio runs into any obstacles or problems. The problem then becomes a question of how to quantify all the wasted work that went into a shot that was not achieved. This whole process is one main contributor to the current lack of financial stability: some companies are overly optimistic in their budget proposals. In creating a more attractive offer, they are underbidding their services by not effectively determining extra costs as a feasible number. Ultimately, it is a problem of old traditions not working in today's environment. (Visual Effects Society, 2010, p. 20)

Things may be looking up for the future of VFX, as industry leaders are addressing this problem. The current model was born circa post-*Star Wars* 1970s, when production companies turned to outside independent companies, rather than hiring individuals to work alongside the production team. Having more independent companies gave the industry more opportunities to work on the research and development sector in order to innovate the VFX in the digital age. We are now at a stage in VFX history where companies do not have to look to invent new technologies, but rather streamline the manufacturing of a standardized type of shot. More work is becoming automated and can be achieved by implementing efficient workflows within the system. Companies that can simplify their work, specialize in something unique, and/or provide a realistic price point will thrive while those that do not improve will face narrower profit margins. (Visual Effects Society, 2013) With time, more companies will adapt to a model that will change the state of the industry for the better.

Problem Statement

In any market, the product value is defined as:

$$\text{Value} = \frac{\text{What you get}}{\text{What you pay}}$$

This may seem like a trivial equation – it may seem obvious that the more you get, and the less you pay, the greater the value of the product becomes. However, it is important to recognize that at the core of any system, the value a customer places on the product is what will pull them in or push them away. (Niku, 2009, p. 383)

As briefly mentioned above, companies with a more efficient system in place will become more profitable than those that do not, and in turn will hold more value. Any small improvements to any of the multiple workflows in the VFX structure (commonly called the VFX pipeline, as a whole) can make a big difference in providing a shot that carries the best value. Taking a look at the pipeline, it is possible to compare it to a number of manufacturing workflows found in the engineering industry. With my senior project, I want to develop a new, unique model that utilizes the engineering techniques found in the engineering industry, to create a VFX workflow that provides the most value.

Criteria for Success: Workflow Considerations

To develop a quality workflow, it is in our best interest to define a standard of what an efficient workflow should entail. In 2012, a paper was published by the people of *Workflow4Ever (Wf4Ever)*, defending steps that allow for higher quality workflow practices and the prevention of workflow decay. In scientific workflows, decay can occur when certain components become obsolete over time, rendering the workflow useless. They attest that the following ten steps impede the rate of decay and can make a workflow easily maintainable (Hettne, et al., 2012):

1. Make an abstract workflow – Create an initial sketch of the workflow. This acts as a means of communicating the goals of the workflow and a reference to its future design.
2. Use modules – Implement executable components into the workflow as separate sub-workflows. This creates a system containing parts that can be easily reused and swapped out if broken.

3. Think about the output – Consider where the output is going to be implemented after going through the workflow.
4. Provide input and output examples – Ensure that the example outputs match the example inputs.
5. Annotate – Record all the steps and assumptions not shown in the workflow. This promises ease of use and re-use of the workflow.
6. Make it executable from outside the local environment* – Look to create the workflow in a global environment. This guarantees portability of the workflow.
   *If local services are the only option, ensure proper annotation of the local tool (i.e. product version number, operating system, licensed or open source, etc.)
7. Choose services carefully – Guarantee status, reliability, and stability of the services found in the workflow.
8. Reuse existing workflow – Work up from currently published methods. There is no need to reinvent the wheel.
9. Test and validate – Include functions in the workflow that can check defined test cases. This certifies the validity of our results.
10. Advertise and maintain – Share your results. Maintain that integrity of your workflow.

Deliverables

Within the VFX pipeline, I will be focusing on the initial animation workflow for a vehicle suspension model. The main goal is to automate a way to animate the vehicle driving over several bumps at a constant velocity. In conjunction with this report, I will create a quick video of the workflow in action and show the process of animating the model from beginning to end. However, as I feel that the animation of the vehicle will add proof of concept for my project to be a success, I will not delve into the rendering of the animation.

Technology Overview

Computer Programs

Autodesk Maya

There are a number of programs that are currently on the market that can be used to simulate and/or animate a vehicle model. I chose to go with Autodesk Maya for its control in animation and its ability to be able to create scripts, thus adding a module component to the workflow. After some research, it appears that although Maya has a steep learning curve, there is a large community that relies on this program, so there is also an ample amount of support for the maintenance of the program.

MathWorks MATLAB

One other third party program utilized in this project is MATLAB, a numerical computation tool used in the engineering workforce. This decision is unorthodox for a typical visual effects project, but the power and versatility for MATLAB to create real life simulations is what drew me to the program. I will be utilizing MATLAB in a similar manner as motion capture: the code will go over each frame of a captured simulation, in order to export over to the model in Maya.

Literature Review

If looking at my project in a broad sense, there isn't much literature on implementing engineering practices into a VFX pipeline. The topic of changing the VFX climate has only come to fruition in recent years, which limits research publications about what possible changes can be implemented and none refer to an engineering model. With that being said there has been one very recent unofficial proposal, made by the design effects company, MAGNOPUS. They proposed looking at the pipeline as a product manufacturing workflow, similar to the one that Apple is applying to its products today.

Apple is one of the most successful companies in the world that creates innovative products for the public. Yet, people tend to overlook the fact that Apple relies on outside suppliers to make their products – Apple technically only owns the schematics to solving their creative problems. This is where MAGNOPUS proposes we follow Apple: we should be able to create a large company that utilizes other small satellite companies that specialize in different aspects of the pipeline. This main company will be able to manage the solving of any creative problems, while the smaller companies focus on the manufacturing of the shoot. In order for this model to work, some companies need to be willing to be transparent within the production world and only act as vendors to the main companies. A "per-unit" type of measurement for billing would then calculate the costs for the vendors. Currently MAGNOPUS is testing out this type of production thinking, and are hoping to spread their research and ideas to other companies, in order to move the industry forward. (Seymour, 2014)

My project is similar looking towards manufacturing workflows to solve our problems. However, where my project differs is that I am looking more into a workflow within the pipeline, rather than the pipeline as a whole. Nonetheless, their opinions are novel in nature and should be used as a reference for my project.

Design Considerations

Lean Production

Taking MAGNOPUS's proposal one-step further, I plan for my deliverable to consider some of the standards found in a specific type of manufacturing: the lean production method. Ideally, I would implement this model into my proposed workflow, but since the model is meant to work alongside collaboration and I am taking on this project independently, I can only outline how I would implement it in future iterations.

Lean Production was first synthesized in 1950 when Eiji Toyoda and Taiichi Ohno set out to change the large volume method of production for their car manufacturing company, Toyota. At the time, Henry Ford had a mass production model in place, focusing on the speed of creating a large number of products in a short amount of time. Lean production differed from this by focusing on the quality output over the quantity of their output. The main attribute that lean production brought to manufacturing was reducing the amount of waste that came out of its assembly line.

One waste reducing technique I found to be most relatable to VFX was the idea that everyone in the assembly line had total control over the final product. For example, if a part were to arrive at a station and cause problems, the whole production would stop until the source of the problem was found. Instant feedback from one another, forced each worker

to fully consider the quality of their work before it was handed to the next person in line. They did not want to be the reason for the whole production to stop. Initially it might have seemed that chaos would have ensued with all these stoppages, but whenever any problems arose, blame would be put on the process, rather than the individual. Thus, attention would be shifted to fixing the workflow rather than blaming the workers. (Niku, 2009, pp. 443-450)

If I had colleagues working on this project alongside me, I would have tested out the practicality of this production model by asking for immediate feedback after each process is completed. If a problem were to arise, we would stop all operations and look at how to improve the workflow. For now, it is simply something I will be keeping in mind when creating the final deliverable, as it is necessary to address the collaborative nature of VFX.

Typical Animation Workflow

There are two types of general animation pipelines: one consisting of entirely animation, and one featuring live-action components for VFX. The overlap between the two pipelines is quite significant, with the exception of the bidding process and pre-production phase.

For animation one singular company filled with animation specialists is often awarded a project. They are given an overall budget focused on the film's specifications, such as length of film, number of set pieces, and overall complexity of the project. This is compared to the VFX animation pipeline, which is equivalent to any other VFX bidding procedure, where multiple companies bid for a shot. In addition, animation quotas are often based on amount of finished work, while VFX animation quotas depend on the quantity of shot counts. These differences could explain why historically, animation companies have prospered more so than their VFX cousins.

Aside from their business models the pre-production phase of each pipeline also differ. Animation has to focus on laying out the scene in 3D space from the storyboard, while VFX focuses more on the process of 'matchmoving' where they translate a real-life 3D space into a computer 3D environment. Before contemporary filmmaking, it was easier to differentiate between a complete animation project and a visual effects project. The earlier description of an animation project having no live-action components attached to it cannot apply to some of today's highly stylized computer generated movies. Because the lack of distinction is still increasing, it is important to understand animation projects even if an animator has previously only worked in the VFX pipeline. (Visual Effects Society, 2010, pp. 737-758)

For the scope of my project, I will be focusing on a simplified model of the animation workflow found in the VFX pipeline, taking some elements from the animation pipeline.

The initial step in creating an animation is to research the main model's real-life look and understand its movement. It is also necessary to match these movements to the environment it will be placed in. One thing to keep in mind during this process is to work within the constraints of the shot, and mimic the real world dynamics, while keeping creative styles in mind. Replicating all the dynamics pieces is not necessary if they are not visible in the shot. (Creative Skillset)

Abstract Workflow

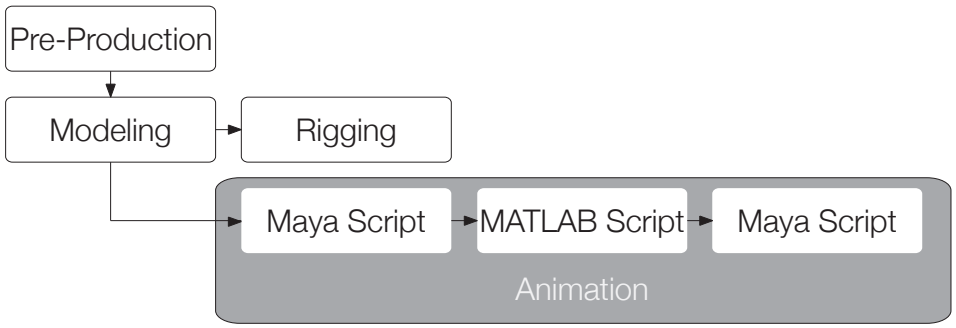


Figure 1: Abstract workflow

Requirements Analysis

In systems engineering, requirements analysis is used to determine the conditions that a system needs to meet in order for the users to achieve a certain goal. For each step in the workflow, it is then necessary to define requirements for the given scenario. Input and output are what goes in and out of the sub-workflow. Functional requirements include what functions will be available to the user, while usability requirements include the ways these functions can be altered. (Visual Effects Society, 2010, pp. 786-791)

Process Overview: Take a 3D mesh, rig and animate the model concurrently until desired movement is achieved
Users: Riggers and animators
Beneficiaries: Riggers, animators, and texture painters

Scenario: Locator tracking in Maya

Input: A mesh with locators placed at key points of simulation.
Functional: Choice of creating a namespace for model.
Curve determining beginning & end global location
Choice of number of frames animation will act in.
Usability: Input prompt for functional parameter tuning.
Automatic renaming of created locators, curves, etc.
Output: File containing x, y, z coordinates, representing Road Profile.

Scenario: Model simulation in MATLAB

Input: File containing x, y, z coordinates, representing Road Profile.
Functional: Varying mass, spring constant, damping constant.
Usability: Input prompt for parameter tuning.
Should include ability to view rough animation of simulation.
Output: File containing 3-D coordinates, representing positions of suspensions.

Scenario: Animating model in Maya

Input: A mesh with locators placed at key points of simulation.
File containing 3-D coordinates, representing positions of suspensions.
Functional: None.
Usability: None.
Output: Animated suspension model, ready for texturing.

Implementation

Initial Research

The first step in creating an animation was to study the real-life comparison of our product animation and understand the physical real-life properties of it. Since I only knew how vehicles operated in a very general sense, I began by researching multiple suspensions systems and watching videos of how they work in real life. All suspension systems have two components to it: the spring and shock absorber. Together, they aim to create a smooth ride for the vehicle – no matter how rough the terrain is, the spring and shock absorber will compress and dampen to make it so that the vertical translation of the car is minimized This is based on a formula dependent on the velocity of the car going into the bump, which I would be inputting into MATLAB. (The Suspension Bible, 2014)

The main suspension system I chose to model was the double wishbone because it is the least complex and could visually demonstrate an exterior suspension system. Also, since my project was mainly focused on the animation, I decided my model would only contain pieces in a vehicle that could exemplify how a suspension system moves over bumps. This meant the animation would only contain the wheels, the frame, and the body.

Primary Iterations

Modeling

The first component modeled was the frame. A cube polygon primitive was created and faces were extruded until there was one half of what I wanted the frame to look like. Once those steps were completed, the geometry was mirrored across the z-axis to create a symmetrical frame. After completing the frame, resolution was added to the polygon with the Insert Edge Loop Tool and the geometry was smoothed out. (digital-tutors, 2014) This frame was then duplicated twice. The original became the upper component of the chassis. One duplicate was scaled up and became the lower component of the chassis. The second duplicate was modified by merging a number of vertices together and changing some of the extrusion variables. This eventually became the body cover.

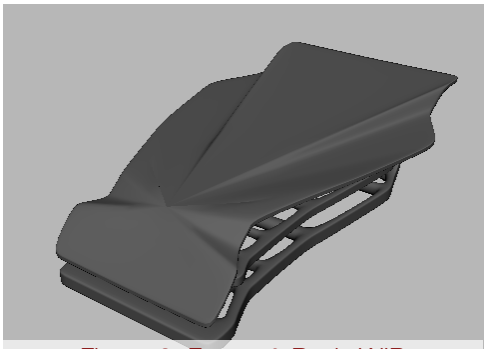


Figure 2: Frame & Body WIP

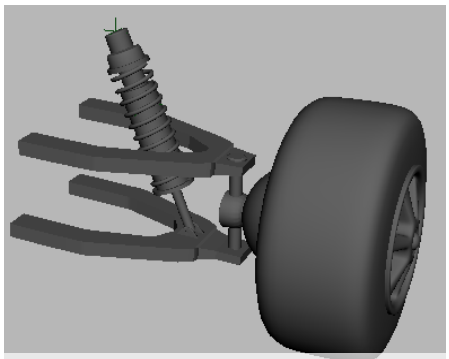


Figure 3: Suspension & Wheel WIP

The next component to model was the suspension / wheel set. The double wishbone joints, for the most part, used the same tools used in making the frame: polygonal modeling, mirroring, and smoothing. To accomplish the wheels, a CV curve was created and revolved about the x-axis to create a NURBS shape. The rims were created with polygon meshes and the bridge tool. (digital-tutors, 2014)

Once all the components were completed, Maya's referencing system was used to import the different components together. The reference tool was a powerful tool to use when importing models that were still being modified because once a change was made in the parent model, that component also changed in the model that was referencing that imported model.

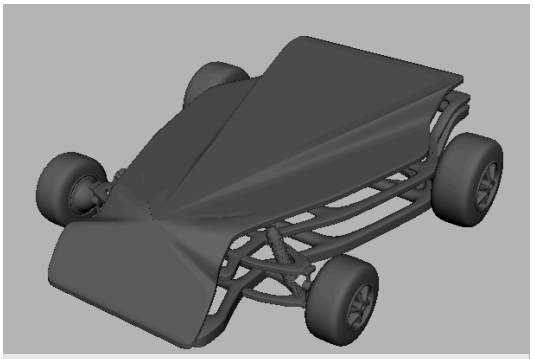


Figure 4: Main Vehicle WIP

Rigging Suspension

My initial strategy to rig and animate my vehicle model was to rig the wheel and have the wheels become the constraints to how the frame and body moves. My final product followed this process, but there were numerous obstacles and iterations along the way.

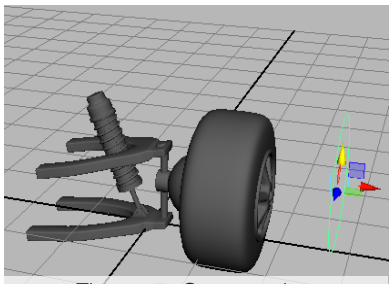


Figure 5: Suspension compressed WIP

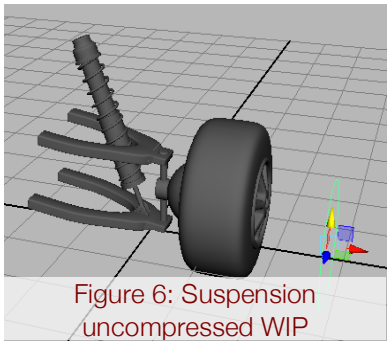


Figure 6: Suspension uncompressed WIP

For the suspension system, point and aim constraints were applied on the upper and lower wishbone, as well as the axles. This allowed the system to follow the motion of the wheel. (Spelce, 2011) The springs surrounding the piston were created using a method found on http://techartandstuff.blogspot.ca. Two helix polygon primitives were created: one representing the compressed spring, and the other of a fully uncompressed spring. They were then rigged using the Blend Shape Deformer, which took the compressed spring and uncompressed spring and combined the two into one shape. The Distance Tool used in conjunction with a SetRange and MultiplyDivide node connection drove the spring rig to compress to difference lengths. (Atkinson, 2013) To put it all together, the movement of the spring was connected to the movement of the wheel using a process known as Set Driven Keys. Set Driven Keys is a convenient tool to drive the motion of an object in terms of a specific attribute of another object, rather than being driven in respect to time. A NURB circle was created and acted as the controller for the wheel. This controller's translation in the Y-axis was then set as the driver for the Set Driven Key sequence and the suspension piston was the object being driven. Therefore, every time the controller moved down, the piston would move up and vice versa, as shown in [Figures 5 & 6].

Animation

My primary solution to solving the animation problem was to track one wheel, and have the rest of the body follow that wheel over the bumps of the road profile. I quickly determined that this was ineffective. The next feasible solution, and the solution that was ultimately used in the final product was to track each

wheel as it traveled from one point to another and have a constraint between the body and each wheel. With this game plan in place, I initially began creating the scripts for the wheels. However, after encountering obstacles with reading the coordinates from a referenced model, I found it to be much smoother to write scripts for a locator and having that locator ultimately become a point constraint for the wheels. This made debugging the code easier because I could reach the source of any problem sooner.

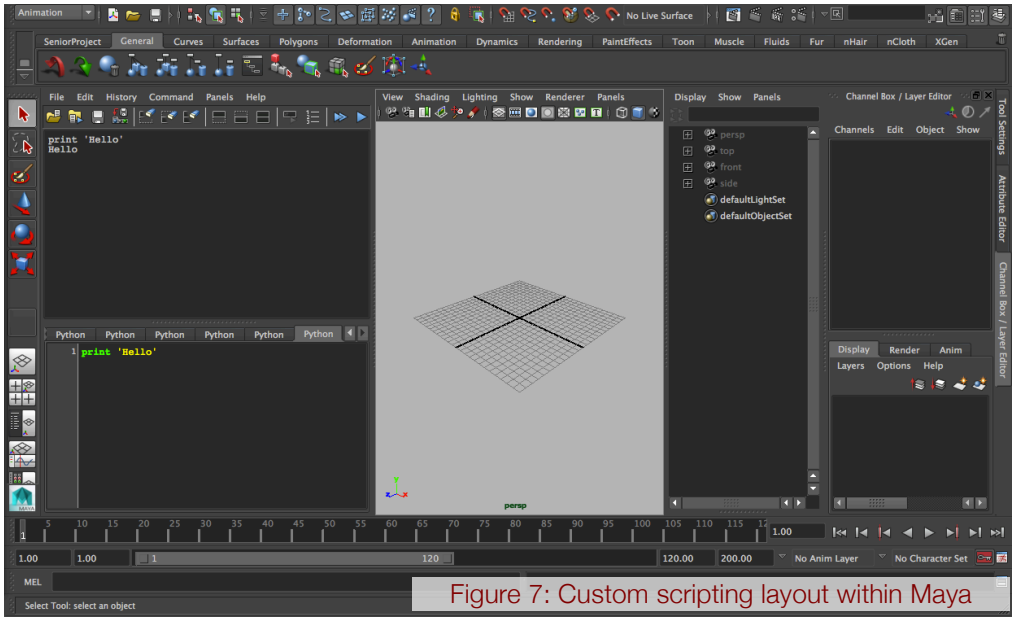


Figure 7: Custom scripting layout within Maya

The following coding scripts were created to animate the locators that would connect to the wheels and suspension:
initial_LOC.py: A locator, named 'vehicle_LOC' is placed onto the roadProfile.

initial_LOCmoPath.py: A motion path is created for 'vehicle_LOC' by first creating a CV curve of the path the wheels will follow and projecting that curve onto the roadProfile mesh.

wheel_LOC.py: Wheel locators are created for each of the wheels on the vehicle and parented under the vehicle_LOC.

wheel_moPath.py: The x and z coordinates of the locator at each frame is read and translated in the y-direction. A motion path is then created for each of the wheels, using the same process as initial_LOCmoPath.py. The script then goes about linearizing the animation curve and reversing any motion paths that are going in the wrong direction.

wheel_writeOut.py: The 3-D coordinates of the locators are read and written out to a .csv file.

suspensionSystem.m: This is the master MATLAB script that calls on the other functions to read and write out our .csv files.

suspension_read_write.m: This MATLAB script reads the .csv file from Maya as a roadProfile, which is inputted into a Simulink model that differentiates an equations of motion file for a simple mass spring damper. It then returns a new .csv file with the suspension coordinates.

wheel_writeln.py: The 3-D coordinates of the suspension are read from the .csv file produced by the MATLAB function. It then goes through each frame of the animation and keyframes the position of the suspension controller.

wheel_ptConstrain.py: A point constraint is put on each wheel by their respective locator

Rigging Frame & Body

Connecting the suspension rig to the frame and body was a more complex anticipated. I was deciding between two types of rigging methods. One was the use of joints, which constrained the movement of the body to the center of the model. The second method consisted of using cluster deformers, which constrained the movement of the body by vertices.

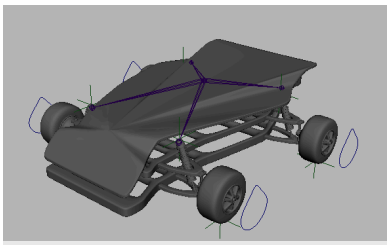


Figure 8: Rigging w/ Joints

For the joint system, the first joint was created on the center of the body, and outlying joints were connected to a locator placed on top of the suspension piston. The joint and body mesh were then connected with the Smooth Bind Tool. This made it so that wherever the suspension piston would move, that section of the body would follow. It was also then possible to use the Paint Weights Tool to determine how much of the body each joint would control, but I did not find it necessary for this rig.

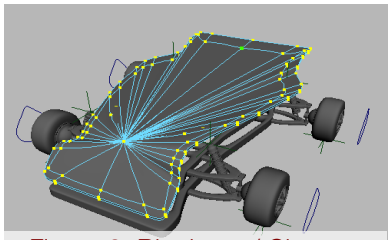


Figure 9: Rigging w/ Clusters

With cluster deformers, it was a more tedious process but offered more precision. Right-clicking the body, I entered vertex mode and selected all of the vertices contained in the body. A cluster deformer for each wheel was then applied to all the vertices selected. With the cluster deformers renamed and selected, the Weight Deformer section in the Component Editor was then used to edit what percentage of what vertex would move when the cluster was translated. For example, the front left suspension would control the front left of the body with a value of 1 and the center of the body with a value of 0.5.

Results

After running into plenty of roadblocks, I arrived at a final product. The model went through minor modifications, mostly to concede to the rigging and animation needs. The suspension rigging went through one major modification, where the motion of the wheel being driven by the controller followed a rotation path about the z-axis. This mirrored a more realistic motion of how a double wishbone suspension should have looked. The adjustment also affected the MATLAB code because the motion path for each wheel locator needed to compensate for the translation in the y-direction that was going on inside the wheels' relative coordinates. As for the body rig, I ultimately decided to go with the cluster deformers because the rigidness of the car could be better shown with this method. Additionally the code was modified to implement more user input options that would prompt the animator of any variables that could be changed.

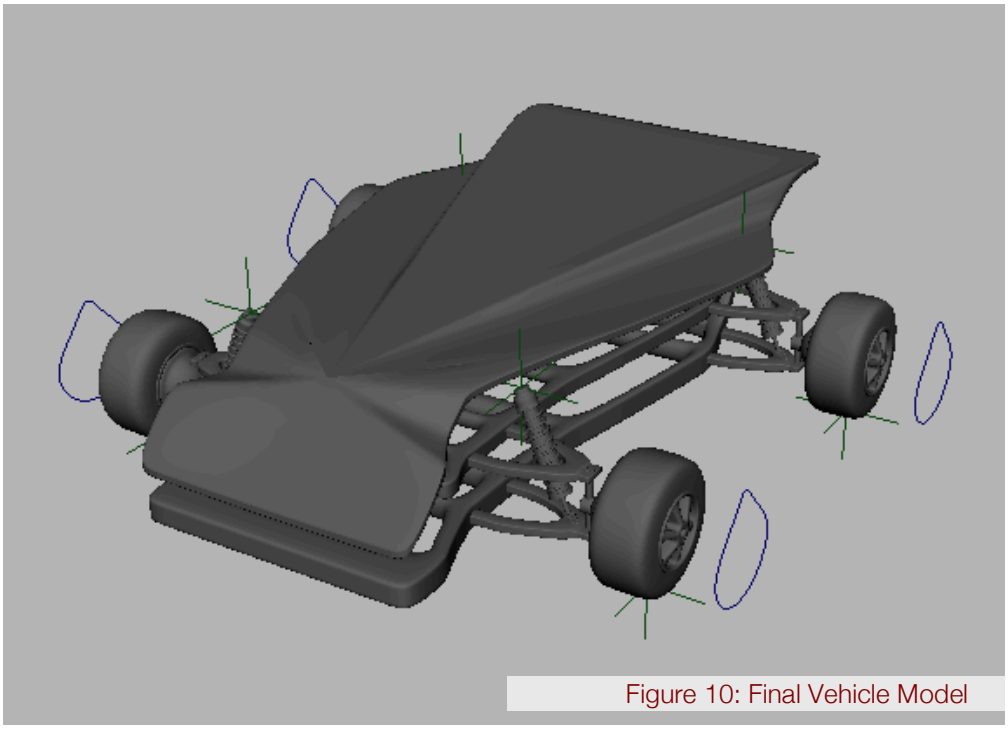


Figure 10: Final Vehicle Model

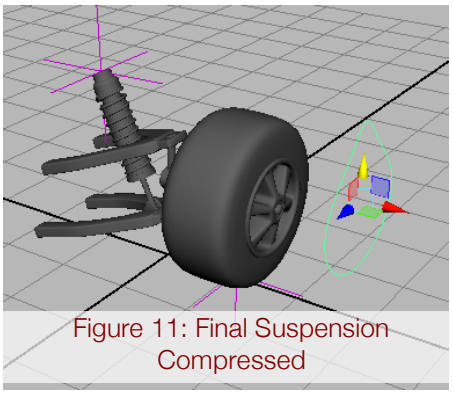
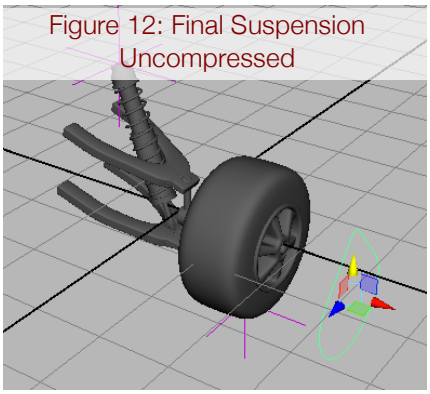


Figure 11: Final Suspension Compressed

Figure 12: Final Suspension Uncompressed



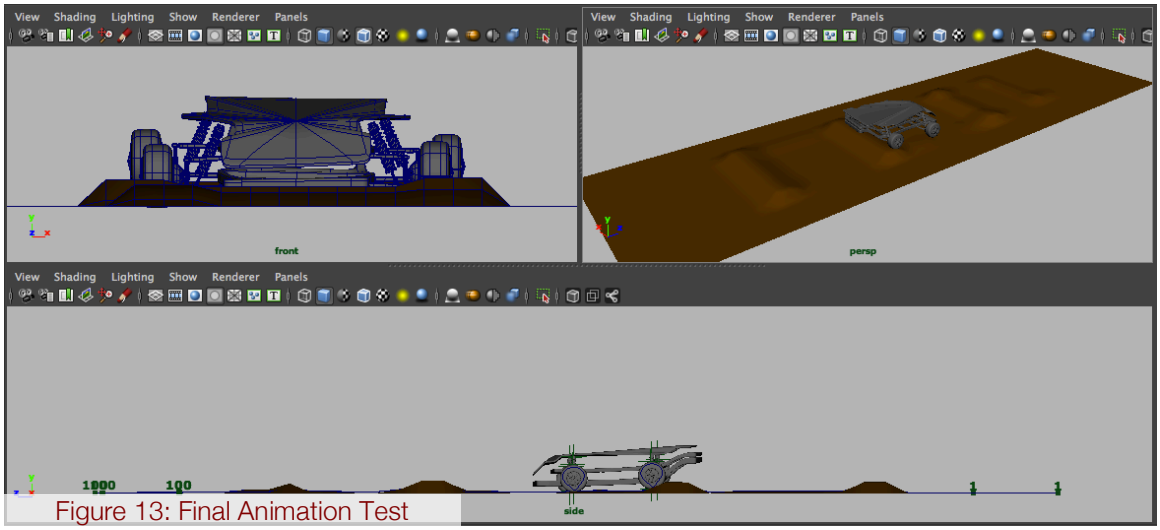


Figure 13: Final Animation Test

With the final workflow created, tests were run where the suspension requirements and terrain were modified. One test sample flipped the default spring and damper setting, and this created a more cartoon-y animation where the suspension system did not compensate for the uneven terrain. Another test sample modified the terrain in which the vehicle would drive over and instead of having to re-keyframing the suspension and wheels of the model to fit the new terrain, the workflow had the suspension system animated within five minutes. These results can be seen on the supplemental video attached to this project. Overall, the relative timeline shown in [Figure 14] shows that with the proposed workflow, there may be more time spent in creating the script modules, but the time it took to animate the model was heavily reduced.

Figure 14: Workflow Comparison

Analysis / Verification

Putting together all of the research and the actual workflow deliverable I would say that my project produced demonstrable results. It is difficult to quantify the results from a project like this into a set of data. Instead, I followed the guidelines to defeat workflow decay, presented by the research done by *Workflow4Ever* and compared my results.

1. Make an abstract workflow – An abstract workflow was created, as well as a requirements analysis in order to define my end goals of what the workflow would look like.
2. Use modules – Maya Python scripts and MATLAB scripts were implemented as modules, which could be modified and replaced as time went on to blend in with new requirement additions.
3. Think about the output – The output was considered throughout both the design and implementation phases of the project.
4. Provide input and output examples – The ability to both read and write .csv files created a synergy between Maya and MATLAB.
5. Annotate – All the steps and assumptions were annotated in this document.
6. Make it executable from outside the local environment* – This was not possible with the local environment in which the programs ran from, but this was not a necessary requirement.
7. Choose services carefully – Maya and MATLAB are both programs that are heavily supported in both Visual Effects and Engineering, respectively. They are consistently maintained and updated.
8. Reuse existing workflow –My proposed workflow had some basis from the lean production model and a typical animation workflow.
9. Test and validate – Tests were ran to see how flexible the workflow was and if any more constraints needed to be applied. Different suspension constants and terrain road profiles were integrated into the model to validate the efficiency of the workflow.
10. Advertise and maintain – The results of my workflow are advertised in this document. I hope to continue to work on the script modules to further the efficiency even more.

Conclusion

Although the state of the visual effects industry is not currently in a great position, there is a movement to change things for the better. The research and application I completed for this project show some suggestions into how the overall model could implement efficient engineering models. Although I do not see my workflow completely phasing out the current process of how things are ran, it is possible to implement alongside current practices and increase the efficiency for certain shots. The downside of implementing MATLAB into an animation workflow is that MATLAB is not a program known by most animators, which is evident by the lack of literature on the combination. If a company were to fully implement MATLAB, they would need to consider the costs of teaching the programming language to their animators.

Future Work

As evident with my project, there are improvements that can come with the inclusion of implementing engineering practices into the visual effects environment. Looking towards the future, I would recommend more flexibility to the MATLAB code (i.e. giving the vehicle lateral functionality and applying an Ackerman Steering System to the vehicle). I would also broaden the scope of this project and reapply this workflow with a collaborative ecosystem and apply manufacturing production techniques along with the module creation. Since VFX is such a collective effort it would be advantageous to understand where a break in the workflow could occur and how to repair that link as soon as possible so as to not create wasted time. There is also the possibility to apply this workflow for other types of animations (i.e. create linkage and gear mechanisms models that rotate in a realistic manner). All in all, this project is just the beginning, and future work can only strengthen what was concluded from this project and hopefully some workflows within the VFX industry.

REFERENCES

Atkinson, P. (2013, June 06). *How To : Rig Accurate Working Springs in Maya*. Retrieved 2014, October 17, from Tech Art and Stuff: http://techartandstuff.blogspot.ca/2013/06/how-to-rig-accurate-working-spring-in.html

Creative Skillset. (n.d.). *The Core Skills of VFX.* Retrieved from http://creativeskillset.org/assets/0000/0236/The_Core_Skills_of_VFX.pdf

digital-tutors. (2014). *Introduction to Modeling in Maya* [Video File]. Retrieved from http://www.digitaltutors.com/tutorial/1151-Introduction-to-Modeling-in-Maya

Leonard, M. (2013). *VFX 102 - History Of Visual Effects* [Video File]. Retrieved from http://www.fxphd.com

Hettne, K., Wolstencroft, K., Belhajjame, K., Goble, C., Mina, E., Dharuri, H., et al. (2012). *Best practices for workflow design: how to prevent workflow decay.* SWAT4LS. Retrieved from http://ceur-ws.org/Vol-952/paper_23.pdf

Niku, S. B. (2009). *Creative Designs of Products and Design.* Hoboken, NJ, USA: John Wiley & Sons, Inc.

Seymour, M. (2014, December 01). *A way forward for the VFX industry*. Retrieved December 5, 2014, from http://www.fxguide.com/featured/a-way-forward-for-the-vfx-industry/

Spelce, K. (2011). *Car Suspension Rigging with Maya* [Video File]. Retrieved from http://www.vimeo.com/20555485

The Suspension Bible. (2014, April 13). Retrieved October 11, 2014, from Car Bibles: http://www.carbibles.com/suspension_bible.html

Visual Effects Society. (2013). *The State of the Global Visual Effects Industry 2013: An Analysis of Current Business Models and Better Business Practices* [White Paper]. Retrieved from http://www.visualeffectssociety.com

Visual Effects Society. (2010). *The VES Handbook of Visual Effects: Industry Standard Practices and Procedures.* (J. A. Okun, & S. Zwerman, Eds.) Burlington, MA, USA: Focal Press.

APPENDIX A: MAYA PYTHON SCRIPTS

#00 Create initial locators

Select FrontLeft // FrontRight // BackLeft // BackRight Wheel Locations

```
import maya.cmds as mc
# determine vehicle number namespace
mc.promptDialog(message='Vehicle number (namespace)',button=['OK'])
vehicle_NUM = str(mc.promptDialog(query=True, text=True))
ns = mc.namespaceInfo(currentNamespace=True)
if ns == ':':
    if mc.namespace(exists='Vehicle' + vehicle_NUM) == 0:
        mc.namespace(add='Vehicle' + vehicle_NUM)
        mc.namespace(set='Vehicle' + vehicle_NUM)
    else:
        mc.namespace(set='Vehicle' + vehicle_NUM)
else:
    mc.namespace(set=':')
    if mc.namespace(exists='Vehicle' + vehicle_NUM) == 0:
        mc.namespace(add='Vehicle' + vehicle_NUM)
        mc.namespace(set='Vehicle' + vehicle_NUM)
    else:
        mc.namespace(set='Vehicle' + vehicle_NUM)

# create list of selected objects
mySel = mc.ls(sl=1)

# assign front left and back right X & Z float values
FL_LOC_X = float(mc.getAttr(mySel[0]+'.worldMatrix')[12])
FL_LOC_Z = float(mc.getAttr(mySel[0]+'.worldMatrix')[14])
BR_LOC_X = float(mc.getAttr(mySel[3]+'.worldMatrix')[12])
BR_LOC_Z = float(mc.getAttr(mySel[3]+'.worldMatrix')[14])

# determine center coordinate
vehicle_LOC_X=(FL_LOC_X + BR_LOC_X) / 2
vehicle_LOC_Z=(FL_LOC_Z + BR_LOC_Z) / 2

# create main vehicle locator
_LOC= mc.spaceLocator(n='vehicle_LOC')
mc.move(vehicle_LOC_X,float(mc.getAttr(mySel[0]+'.worldMatrix')[13]),vehicle_
LOC_Z)

# create destination vehicle locator
_LOC= mc.spaceLocator(n='end_vehicle_LOC')
mc.move(vehicle_LOC_X,float(mc.getAttr(mySel[0]+'.worldMatrix')[13]),0)
mc.setAttr(ns+':end_vehicle_LOC.translateX', lock=1)
mc.setAttr(ns+':end_vehicle_LOC.translateY', lock=1)
```

APPENDIX A: MAYA PYTHON SCRIPTS

#01 Motion path for initial locators

```
import maya.cmds as mc
# determine number of frames
mc.promptDialog(message='Number of frames',button=['OK'])
frames = int(mc.promptDialog(query=True, text=True))

# identify beginning & end locator X & Z float values
_LOC1X = float(mc.getAttr(ns+':vehicle_LOC.translateX'))
_LOC1Z = float(mc.getAttr(ns+':vehicle_LOC.translateZ'))
_LOC2X = float(mc.getAttr(ns+':end_vehicle_LOC.translateX'))
_LOC2Z = float(mc.getAttr(ns+':end_vehicle_LOC.translateZ'))
# create curve with coords array
mc.curve( n='initialCRV', d=1, p=[(_LOC1X,10,_LOC1Z),(_LOC2X,10,_LOC2Z)] )
# project selected curve on selected surface
mc.polyProjectCurve(ns+':initialCRV', 'roadProfile', n='vehicle_CRV', d=(0.0,
1.0, 0.0), ch=0)
mc.delete(ns+':initialCRV')
#put locator on path
mc.pathAnimation(ns+':vehicle_LOC', curve=ns+':vehicle_CRVShape',
n='vehicle_moPath', startTimeU=1, endTimeU=frames)

# reverse curve if direction of motion path is reversed
# enter first frame
mc.currentTime(1)
# current locator translateX & translateZ values
location_Matrix = mc.getAttr(ns+':vehicle_LOC.worldMatrix')
location_X = int(location_Matrix[12])
location_Z = int(location_Matrix[14])
# if current locator translation does not equal original
if int(_LOC1X) != location_X or int(_LOC1Z) != location_Z:
   # reverse curve direction
   mc.reverseCurve(ns+':vehicle_CRVShape', ch=1, rpo=1)

# delete end vehicle locator
mc.delete(ns+':end_vehicle_LOC')
# hide vehicle crvshape
mc.hide(ns+':vehicle_CRVShape')
```

APPENDIX A: MAYA PYTHON SCRIPTS

#02 Create wheel locators

Select FrontLeft // FrontRight // BackLeft // BackRight Wheel Locations

```
import maya.cmds as mc
# create list of selected objects
mySel = mc.ls(sl=1)

def whl_LOC(_LOC):
    import maya.cmds as mc

    # switch cases
    if _LOC == 'fr_left_Wheel_LOC':
        i = 0
    elif _LOC == 'fr_rght_Wheel_LOC':
        i = 1
    elif _LOC == 'bk_left_Wheel_LOC':
        i = 2
    elif _LOC == 'bk_rght_Wheel_LOC':
        i = 3

    # Find initial position of vehicle_LOC
    wheel_LOC_X = float(mc.getAttr(mySel[i]+'.worldMatrix')[12])
    wheel_LOC_Y = float(mc.getAttr(mySel[i]+'.worldMatrix')[13])
    wheel_LOC_Z = float(mc.getAttr(mySel[i]+'.worldMatrix')[14])

    # Insert locators for wheels
    _LOC= mc.spaceLocator(n=_LOC)
    mc.move(wheel_LOC_X,wheel_LOC_Y,wheel_LOC_Z)

    # parent locators under main vehicle locator
    mc.parent(_LOC,ns+':vehicle_LOC')

whl_LOC('fr_left_Wheel_LOC')
whl_LOC('fr_rght_Wheel_LOC')
whl_LOC('bk_left_Wheel_LOC')
whl_LOC('bk_rght_Wheel_LOC')
```

APPENDIX A: MAYA PYTHON SCRIPTS

#03 Motion path for wheels

```
def moPathCRV(_LOC,_CRV,_SRF):
    import maya.cmds as mc
    # create empty coords array
    coords = []
    # for loop to call world translate of locator
    for i in range (0, frames):
        mc.currentTime( i, edit=True )
        location_Matrix = mc.getAttr(ns+':'+_LOC + '.worldMatrix')
        location_X = float(location_Matrix[12])
        location_Z = float(location_Matrix[14])
        # insert translate tuple into coords array
        coords.append( (location_X, 10, location_Z) )
    # create curve with coords array
    mc.curve( n=_CRV, d=1, p=coords)
    # delete locator
    mc.delete(ns+':'+_LOC)
    # project curve onto polygon plane surface
    mc.polyProjectCurve(ns+':'+_CRV, _SRF, n=_CRV, d=(0.0, 1.0, 0.0), ch=0)
    # create locator
    mc.spaceLocator(n=_LOC)
    # create path animation with newly created locator
    mc.pathAnimation(ns+':'+_LOC, curve=ns+':'+_CRV+'Shape',
    n=_CRV+'_moPath', stu=1, etu=frames)
    # delete curve history
    mc.delete(ns+':'+_CRV)
    # linearize motion paths
    mc.keyTangent(ns+':'+_CRV + '_moPath_uValue', itt='linear', ott='linear')

    # reverse curve if direction of motion path is reversed
    # enter first frame
    mc.currentTime(1)
    # current locator translateX & translateZ values
    location_Matrix = mc.getAttr(ns+':'+_LOC + '.worldMatrix')
    location_X = int(location_Matrix[12])
    location_Z = int(location_Matrix[14])
    # if current locator translation does not equal original
    if int(coords[0][0]) != location_X or int(coords[0][2]) != location_Z:
        # reverse curve direction
        mc.reverseCurve(ns+':'+_CRV+'Shape', ch=1, rpo=1)

moPathCRV('fr_left_Wheel_LOC','fr_left_CRV','roadProfile')
moPathCRV('fr_rght_Wheel_LOC','fr_rght_CRV','roadProfile')
moPathCRV('bk_left_Wheel_LOC','bk_left_CRV','roadProfile')
moPathCRV('bk_rght_Wheel_LOC','bk_rght_CRV','roadProfile')

# delete vehicle locator
   mc.delete(ns+':vehicle_LOC',ns+':vehicle_CRVShape')
```

APPENDIX A: MAYA PYTHON SCRIPTS

#04 Write out road profile coordinates to .csv file

```
import maya.cmds as mc

# determine file location
file_loc = str(mc.fileDialog2(fm=3)[0])

# write coordinates into csv file
def getCOORDS(_LOC):
    import maya.cmds as mc
    import csv
    global file_loc
    # open a file for writing
    getLOC = open(file_loc + '/' + _LOC + '.csv', 'wb')
    # create csv writer object
    mywriter = csv.writer(getLOC)
    # for loop writing out position
    for i in range(0, frames):
        mc.currentTime( i, edit=True )
        location_Matrix = mc.getAttr(ns+':'+_LOC + '.worldMatrix')
        location_X = location_Matrix[12]
        location_Y = location_Matrix[13]
        location_Z = location_Matrix[14]
        mywriter.writerow([str(location_X), str(location_Y),
str(location_Z)])
    getLOC.close()

getCOORDS('fr_left_Wheel_LOC')
getCOORDS('fr_rght_Wheel_LOC')
getCOORDS('bk_left_Wheel_LOC')
getCOORDS('bk_rght_Wheel_LOC')
```

#05 Write in suspension coordinates from .csv file

Select FrontLeft // FrontRight // BackLeft // BackRight Suspension Controllers

```
import maya.cmds as mc

# determine file location
file_loc = str(mc.fileDialog2(fm=3)[0])

# create list of selected objects
mySel = mc.ls(sl=1)

def readCOORDS(_WHL):
    import maya.cmds as mc
    global file_loc
    # read suspension control coordinates
    readSUSP = open(file_loc + '/' + _WHL + '_Wheel_SUSP.csv', 'rU')
    # create empty translate array
    translateSUSP_Y = []
    # append tuples into array
    for line in readSUSP:
        column = line.split(',')
        translateSUSP_Y.append((column[1]))
    readSUSP.close()

    # read new locator coordinates
    readLOC = open(file_loc + '/' + _WHL + '_Wheel_LOC_new.csv', 'rU')
    # create empty translate arrays
    translate_X = []
    translate_Y = []
    translate_Z = []
    # append tuples into array
    for line in readLOC:
        column = line.split(',')
        translate_X.append((column[0]))
        translate_Y.append((column[1]))
        translate_Z.append((column[2]))
    readLOC.close()

    # delete old locator & old locator curve
    mc.delete(ns+':'+_WHL+'_Wheel_LOC',ns+':'+_WHL+'_CRVShape')

    # for loop to animate world translate of suspension and locator
    coords = []
    for i in range(0, frames):
        mc.currentTime( i, edit=True )
        # suspension keyframes
        location_Y = float(translateSUSP_Y[i])
        if _WHL == 'fr_left':
            mc.setKeyframe( mySel[0], at='translateY', v=location_Y, t=i )
        elif _WHL == 'fr_rght':
            mc.setKeyframe( mySel[1], at='translateY', v=location_Y, t=i )
        elif _WHL == 'bk_left':
            mc.setKeyframe( mySel[2], at='translateY', v=location_Y, t=i )
        elif _WHL == 'bk_rght':
            mc.setKeyframe( mySel[3], at='translateY', v=location_Y, t=i )
```

```
            # new locator keyframes
            location_X = float(translate_X[i])
            location_Y = float(translate_Y[i])
            location_Z = float(translate_Z[i])
            coords.append( (location_X, location_Y, location_Z) )

        # create curve with coords array
        mc.curve( n=_WHL+'_CRV', d=2, p=coords)
        # create locator
        mc.spaceLocator(n=_WHL+'_Wheel_LOC')
        # create path animation with newly created locator
        mc.pathAnimation(ns+':'+_WHL+'_Wheel_LOC', curve=ns+':'+_WHL+'_CRV',
        n=_WHL+'_moPath', stu=1, etu=frames)
        # linearize motion paths
        mc.keyTangent(ns+':'+ _WHL + '_moPath_uValue', itt='linear',
        ott='linear')
        # reverse curve if direction of motion path is reversed
        # enter first frame
        mc.currentTime(1)
        # current locator translateX & translateZ values
        location_Matrix = mc.getAttr(ns+':'+ _WHL+'_Wheel_LOC' + '.worldMatrix')
        location_X = int(location_Matrix[12])
        location_Z = int(location_Matrix[14])
        # if current locator translation does not equal original
        if int(coords[0][0]) != location_X or int(coords[0][2]) != location_Z:
            # reverse curve direction
            mc.reverseCurve(ns+':'+ _CRV+'Shape', ch=1, rpo=1)

readCOORDS('fr_left')
readCOORDS('fr_rght')
readCOORDS('bk_left')
readCOORDS('bk_rght')
```

APPENDIX A: MAYA PYTHON SCRIPTS

#06 Point constrain wheels to locators

Select FrontLeft // FrontRight // BackLeft // BackRight Wheels

```
import maya.cmds as mc

# enter first frame
mc.currentTime(1)

# create list of selected objects
mySel = mc.ls(sl=1)

# create point constraints between wheel and locator
mc.pointConstraint(ns+':'+'fr_left_Wheel_LOC', mySel[0], mo=1)
mc.pointConstraint(ns+':'+'fr_rght_Wheel_LOC', mySel[1], mo=1)
mc.pointConstraint(ns+':'+'bk_left_Wheel_LOC', mySel[2], mo=1)
mc.pointConstraint(ns+':'+'bk_rght_Wheel_LOC', mySel[3], mo=1)

# set namespace back to default
mc.namespace(set=':')
```

APPENDIX B: MATLAB SCRIPTS

Suspension System Master File

```
%% Senior Project Suspension System
% Charles Duong

%% Clear the playing field
clc
clear
close all

%% Suspension coordinates read in / write out
global m c k totalTime frames timeStep v myRoad

prompt1 = 'mass (m)? [kg]';                         % prompt user input - mass
m = input(prompt1);                                 % [kg]
if isempty(m)
    m = 500;                                        % default value = 500
end

prompt2 = 'damping coefficient (c) [N-s/m]';        % prompt user input - damping
c = input(prompt2);                                 % [N-s/m]
if isempty(c)
    c = 500;                                        % default value = 500
end

prompt3 = 'spring coefficient (k) [N/m]';           % prompt user input - spring
k = input(prompt3);                                 % [N/m]
if isempty(k)
    k = 20;                                         % default value = 500
end

suspension_read_write('fr_left_Wheel_LOC.csv','fr_left_Wheel_SUSP.csv',1);
suspension_read_write('fr_rght_Wheel_LOC.csv','fr_rght_Wheel_SUSP.csv',2);
suspension_read_write('bk_left_Wheel_LOC.csv','bk_left_Wheel_SUSP.csv',3);
suspension_read_write('bk_rght_Wheel_LOC.csv','bk_rght_Wheel_SUSP.csv',4);


%% New locator coordinates read in / write out
global upperLimit lowerLimit zeroLimit

upperLimit = 2.349;      % Y-coordinate of wheel when wheel_CTRL = 1
lowerLimit = 3.215;      % Y-coordinate of wheel when wheel_CTRL = -1
zeroLimit = 2.788;       % Y-coordinate of wheel when wheel_CTRL = 0

locator_read_write('fr_left_Wheel_LOC.csv','fr_left_Wheel_SUSP.csv',...
    'fr_left_Wheel_LOC_new.csv');
locator_read_write('fr_rght_Wheel_LOC.csv','fr_rght_Wheel_SUSP.csv',...
    'fr_rght_Wheel_LOC_new.csv');
locator_read_write('bk_left_Wheel_LOC.csv','bk_left_Wheel_SUSP.csv',...
    'bk_left_Wheel_LOC_new.csv');
locator_read_write('bk_rght_Wheel_LOC.csv','bk_rght_Wheel_SUSP.csv',...
    'bk_rght_Wheel_LOC_new.csv');
```

APPENDIX B: MATLAB SCRIPTS

Suspension Coordinates Read In / Write Out

```
function suspension_read_write(LOC_name, SUSP_name, plot_NUM)
global m c k frames totalTime timeStep v myRoad zeroLimit

%% Obtain a ground profile
myRoad_raw = csvread(LOC_name); % read in road profile

X_dist = hypot(myRoad_raw(2,1)-myRoad_raw(1,1), myRoad_raw(2,3)-
myRoad_raw(1,3));

i = 2;
while i < length(myRoad_raw)-1  % while loop - determine total distance x-
axis
    X_dist = X_dist + ...
        hypot(myRoad_raw(i+1,1)-myRoad_raw(i,1), myRoad_raw(i+1,3)-
myRoad_raw(i,3));
    i = i + 1;
end

frames = length(myRoad_raw);     % [# of frames]
totalTime = frames/24;           % [sec]
v = X_dist/totalTime;            % [X_dist per second]

myRoad = zeros(length(myRoad_raw),2);   % create empty myRoad array
myRoad(:,1) = linspace(0,X_dist,length(myRoad_raw))'; % road profile x-coords
myRoad(:,2) = myRoad_raw(:,2);  % road profile y-coords

%% Simulink block diagram
timeStep = totalTime/frames;     % defined timestep
sim('Suspension_Simulation');    % run simulink file

%% Road Profile Plot // Suspension Animation
figure(1)                        % Rough animation of suspension
t=0:timeStep:totalTime;          % [sec]
for i = 1:length(simout)         % Loop over the solution data

    x(i) = simout(i,1);             % current wheel x-coord
    y(i) = simout(i,2);             % current wheel y-coord
    z(i) = myRoad(1,2) + zeroLimit + simout(i,4); % current susp y-coord

    plot(x(i), y(i), 'k-o',...              % plot current wheel y-coord
        x(i), z(i), 'k-o',...               % plot current suspension y-coord
        [x(i) x(i)], [y(i) z(i)], 'k',...   % connect wheel and susp y-coord
        simout(:,1), simout(:,2), 'b');     % plot road profile
    axis([x(i)-2.5 x(i)+2.5 myRoad(1,2) 7]);      % define axis
    pause(max(t)/length(t));    % pause for 1 / frame rate
end

figure(2);                       % plot road vs. suspension location (real)
title('Road vs. Suspension')
subplot(4,1,plot_NUM);           % create subplot for plot
title(LOC_name);
hold on
plot(x(:), z(:), 'k',...         % suspension plot
    simout(:,1), simout(:,2), 'b'); % road plot
hold off
```

```
%% Export suspension coordinates to csv file
suspensionY = simout(:,4);      % call on suspension y-coord
for i = 1:length(myRoad)        % modify suspension coords to match rig
    suspDIST(i,1) = myRoad(i,2) - suspensionY(i) - myRoad(1,2);
end

figure(3);                      % plot road vs. suspension location (Maya)
title('Road vs. Suspension (Maya)');
subplot(4,1,plot_NUM);
title(LOC_name);
plot(1:frames,suspDIST,'k',...  % suspension curve for Maya
    1:frames,myRoad(:,2));      % road curve for Maya

suspensionCOORDs(:,1) = myRoad_raw(:,1);    % write out suspension x-coord
suspensionCOORDs(:,2) = suspDIST(:,1);      % write out suspension y-coord
suspensionCOORDs(:,3) = myRoad_raw(:,3);    % write out suspension z-coord
csvwrite(SUSP_name, suspensionCOORDs)       % write out susp coords to csv

end
```

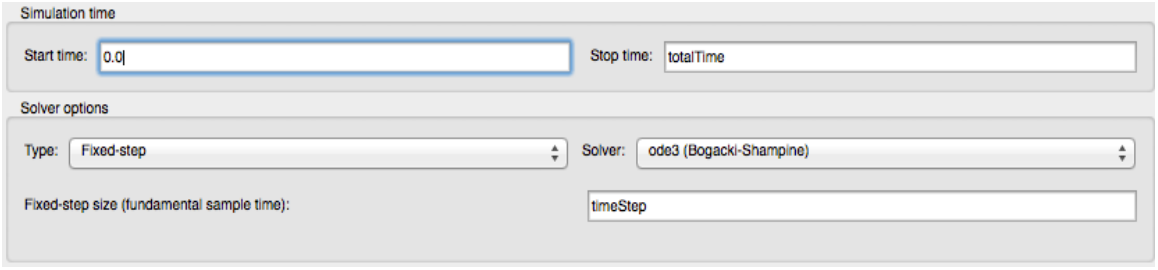
APPENDIX B: MATLAB SCRIPTS

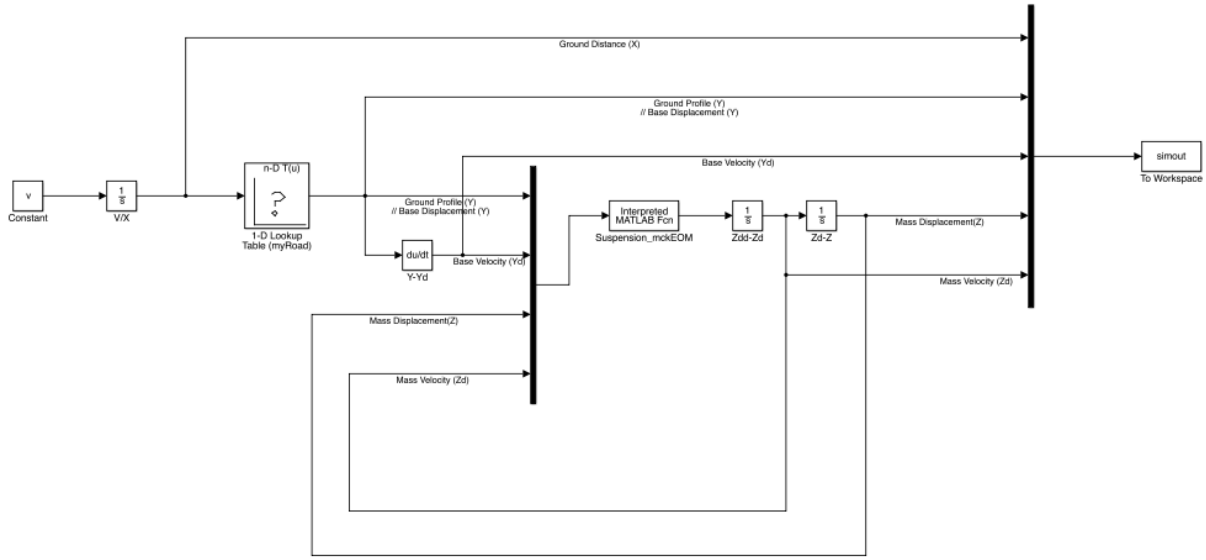
Locator Read In / New Locator Write Out

```
function locator_read_write(LOC_name, SUSP_name, LOC_NEW)
global frames
global upperLimit lowerLimit zeroLimit

%% Obtain a profile
myRoad_raw = csvread(LOC_name);      % read in road profile
mySusp_raw = csvread(SUSP_name);     % read in suspension coordinates

%% Array of y-Coords
translateY(:,1) = myRoad_raw(:,2);   % Y-coords of road profile
translateY(:,2) = mySusp_raw(:,2);   % Y-coords of suspension

%% Set Range
upperRange = zeroLimit-upperLimit;   % set range b/w CTRL.TranslateY = 1 & 0
lowerRange = lowerLimit-zeroLimit;   % set range b/w CTRL.TranslateY = -1 & 0

for i = 1:length(translateY)         % loop to find new locator Y-coord
    range = translateY(i,2)-translateY(1,2);    % total range b/w -1 & 1
    if range > 0                     % if range in upper limit
        transY_new(i) = translateY(i,1) - range*upperRange;
    else                             % if range in lower limit
        transY_new(i) = translateY(i,1) - range*lowerRange;
    end
end

figure(4)   % plot old wheel locator coords vs new wheel locatos
plot(1:frames,transY_new,'k',...     % new wheel curve
    1:frames,translateY(:,1));       % old wheel curve

%% Write new locator csv
LOC_new(:,1) = myRoad_raw(:,1);      % write out new locator x-coord
LOC_new(:,2) = transY_new;           % write out new locator y-coord
LOC_new(:,3) = myRoad_raw(:,3);      % write out new locator z-coord
csvwrite(LOC_NEW, LOC_new)           % write out suspension coords to csv

end
```

APPENDIX B: MATLAB SCRIPTS

Simulink Diagram (Suspension_Simulation)





Suspension Equations of Motion (EOM) File

```
function zdd = Suspension_mckEOM(u)
global m c k
zdd = (1/m) * (  k*(u(1)-u(3)) + c*(u(2)-u(4)) );
end
```