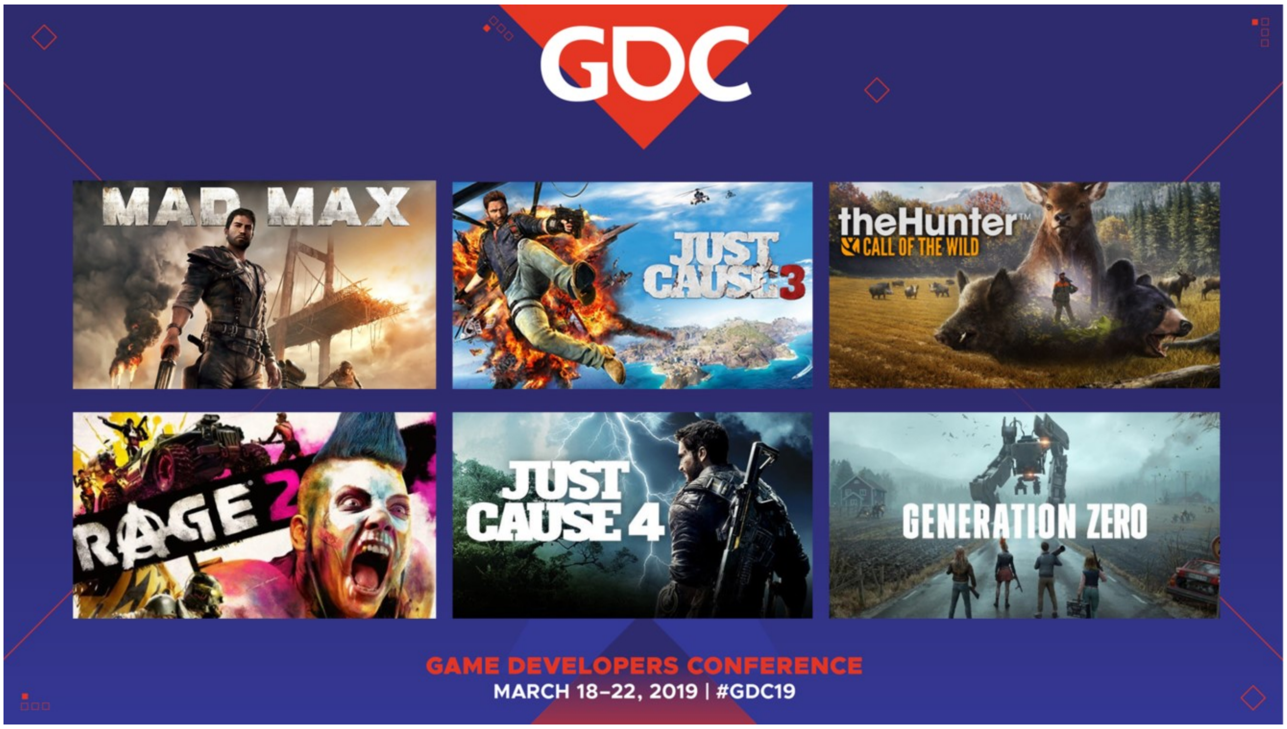
Maximizing Animation & Cinematic Content Workflows for Just Cause 4

Brian Venisky
Senior Technical Animator – Avalanche Studios
Twitter: @TechAnimator

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

Hi everyone, I'm Brian Venisky and I'm a Senior Technical Animator at Avalanche Studios. I'm located…

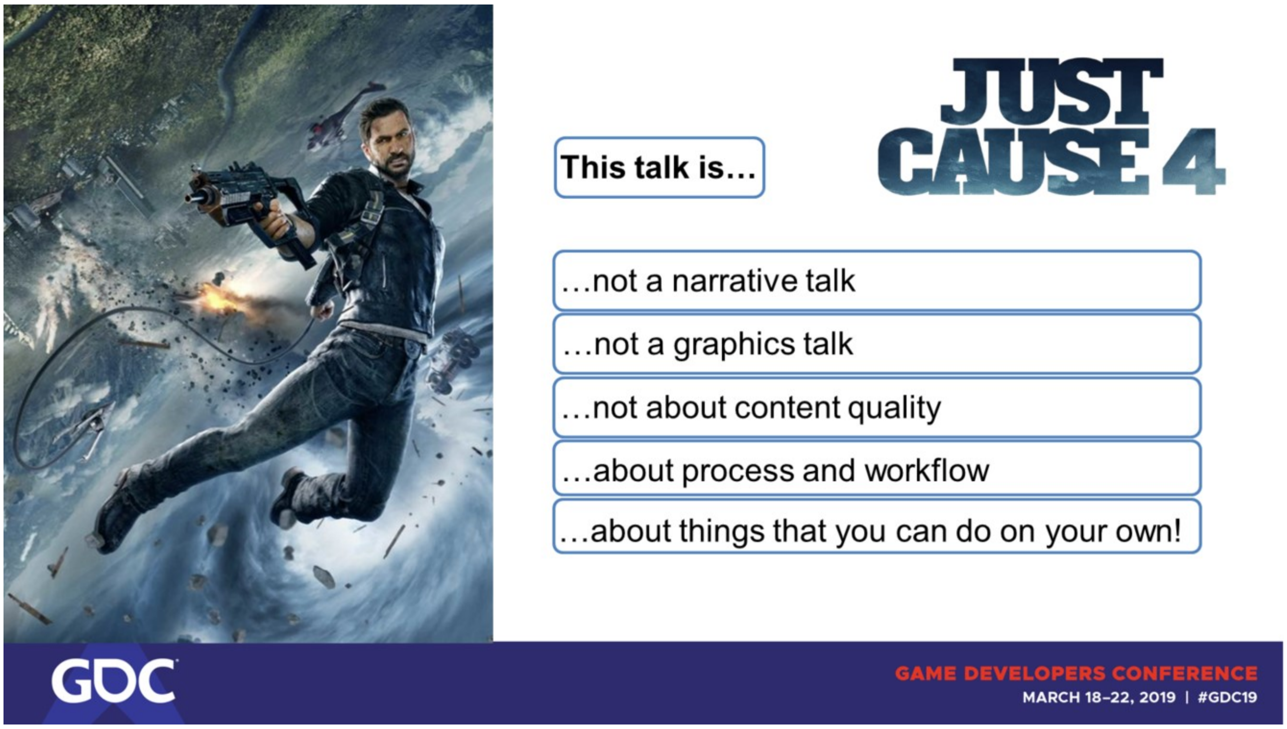
…in the NYC office right in the middle of Manhattan…with our other 2 locations both being in Sweden. There's our OG studio in Stockholm and then our newest studio, which officially opened last year, in Malmo.


GDC
MAD MAX
JUST CAUSE 3
theHunter CALL OF THE WILD
RAGE 2
JUST CAUSE 4
GENERATION ZERO
GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

Our most recent games you may know about are the latest installments in the Just Cause series of course, with Just Cause 4 having been released this past December, Mad Max, Rage2 which comes out on May 14 (2019) and notable self published games with theHunter Call of the wild as well as Generation Zero which comes out THIS Tuesday March 26 (2019)!

Today, I'll be talking about the content workflows that we've employed for both the gameplay and cinematic teams that began from the ground up during Just Cause 3 and evolved over the production of Just Cause 4.

The reason why I wanted to talk about the work we've been doing over the last few years is because I truly believe that what we've been able to do with the scope of our projects and the team size we've had is quite remarkable. It's not only a testament to the tools and processes that have been employed, but by the fantastic teams that I've had the pleasure of working with.



This talk is…

JUST CAUSE 4

…not a narrative talk

…not a graphics talk

…not about content quality

…about process and workflow

…about things that you can do on your own!

GDC

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

I would like to set the expectations going into this to better help you all understand what I'm going to talk about.

Now, consumers look at your product and they see the shiny things. They rate your graphics, how you compare to other games, and so on. But at the core of it all, we are producing content in very similar ways, using workflows and methods underneath that shiny finish.

A large focus of mine over the years has involved a lot of process and with that said, I'm not going to give you a groundbreaking new piece of tech during this talk as it is in fact about using and manipulating things that already exist. And while I love sharing my thoughts and process, I'm not saying this is all the definitive way to do something, but it worked well for us.

While a good portion of this talk is focused on cutscene work, it is not a narrative talk so I won't be going into detail about that side of cinematics, although certain parts of our pipeline were created to allow for the extra work that came into narrative changes.

It's also not a talk about the overall look of anything or the in-game tech behind it.

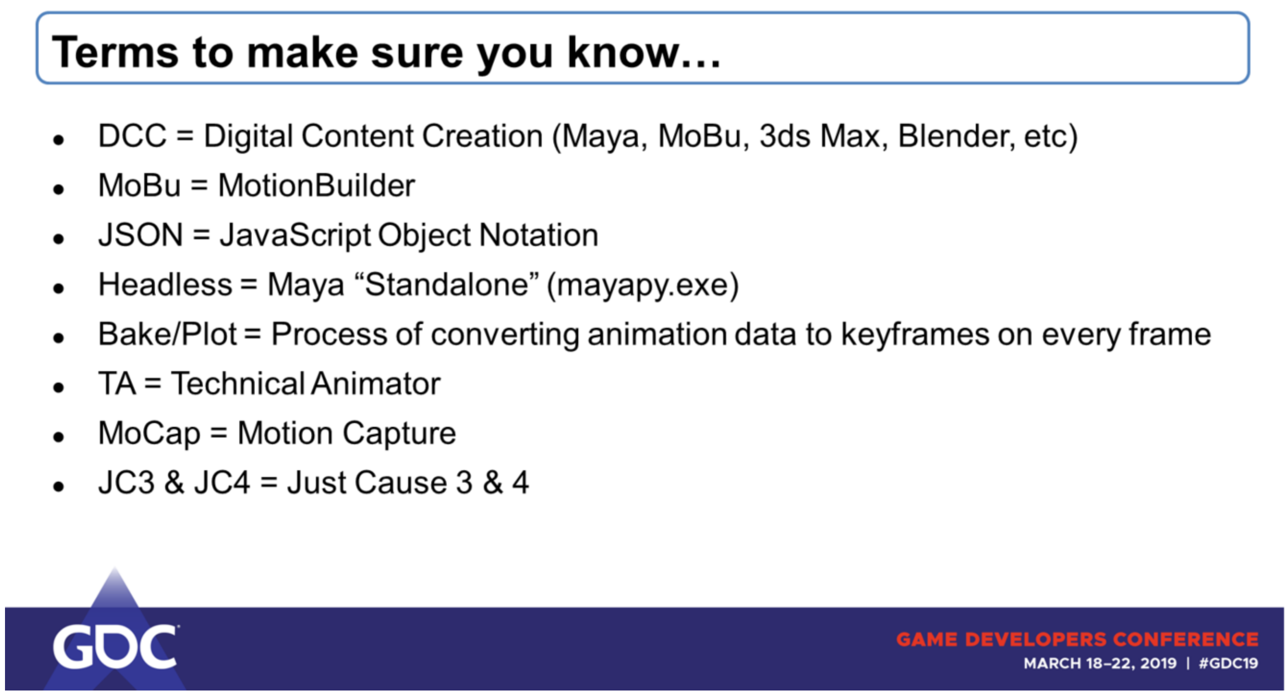
And It's not a talk that discusses creating high quality content or how to hit a certain level of quality.

It IS about managing large quantities of content and making the lives of the animators a little easier through decisions and work done to improve the process of content creation during production of Just Cause 4.

And I wanted to make sure that no matter the position you're in, whether it be a small studio, large studio, or just yourself…that you can take a few things back with you from this and without the need of a programmer or some special tech, do any of the things that I've presented on your own as you don't always have the luxury of programming support or an army of tech artists or tech animators.

I also wanted to make sure that this talk was accessible for any level of knowledge. Because of this I'll probably go over things that some of you already know or have done yourselves, in order to better help out those who may not know about particular topics. This is even more prominent when I get into MotionBuilder specifics because of the greater lack of support and knowledge that you find out there compared to Maya, which has far more information to be found as quickly and easily as typing any broad topic into a search engine.

Slight heads up, this talk is about to be a bit of a hodge-podge because I have some odds and ends that I want to share amongst some of our bigger tasks, so hopefully you don't mind a bit of random tips and/or tricks mixed in with pipeline and workflow.



Terms to make sure you know…

- DCC = Digital Content Creation (Maya, MoBu, 3ds Max, Blender, etc)
- MoBu = MotionBuilder
- JSON = JavaScript Object Notation
- Headless = Maya “Standalone” (mayapy.exe)
- Bake/Plot = Process of converting animation data to keyframes on every frame
- TA = Technical Animator
- MoCap = Motion Capture
- JC3 & JC4 = Just Cause 3 & 4

GDC GAME DEVELOPERS CONFERENCE MARCH 18-22, 2019 | #GDC19

So in the effort of keeping this talk as accessible as possible, I want to define some terms because I’ve found that different disciplines tend to not know some specific shorthand verbiage.

DCC refers to digital content creation application and for this talk the DCCs that I’ll be mentioning are Maya and MotionBuilder.

MoBu is shorthand for MotionBuilder.

JSON stands for JavaScript Object Notation, but simply put it’s a language independent data format that lets you store data to be accessed in a very straightforward and logical manner. If you’re aware of what a dictionary is in coding, json looks and behaves very similarly on the surface.

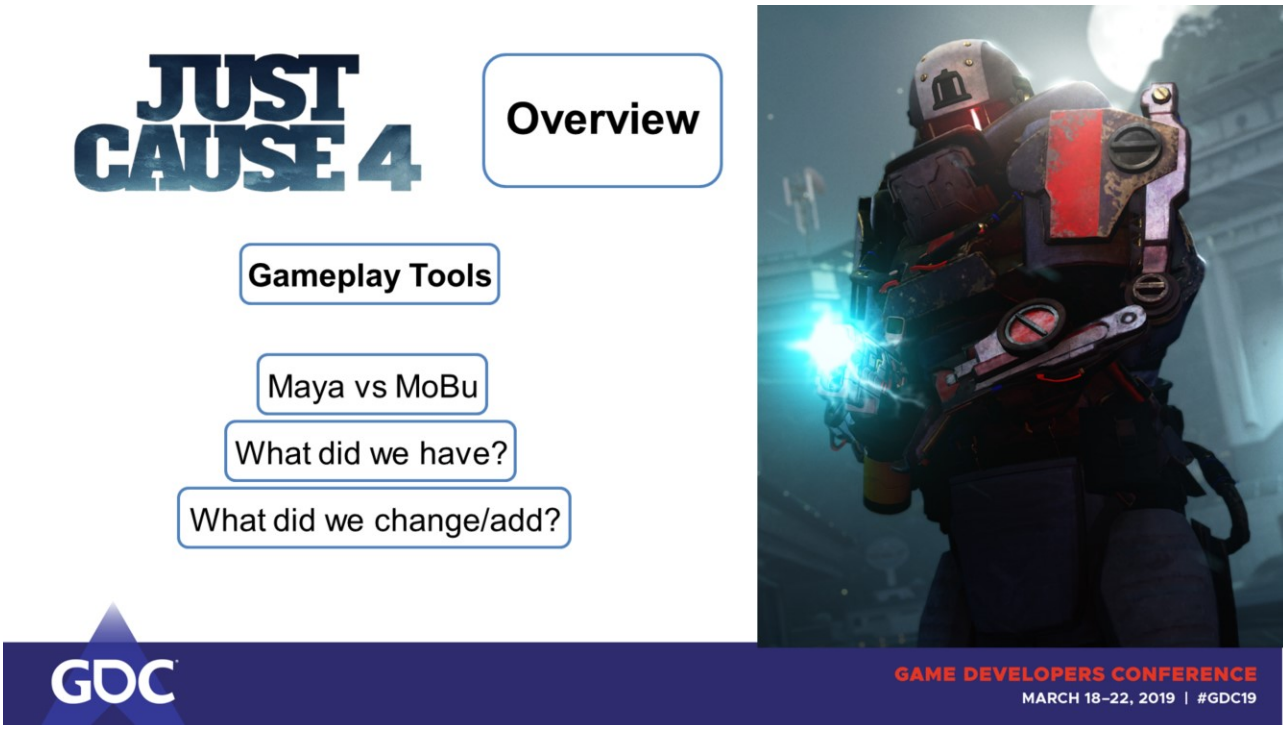
Headless refers to running maya in standalone mode which means you’re using full on Maya and all of it’s features, but it runs in the background without actually opening up it’s visual component.

Baking and Plotting as animators and tech animators should already know is the process of converting animation data to keyframes on every frame. Since this term is different between applications, I want to make sure everyone knows that baking is the term for this process in Maya, and plotting is the term in MotionBuilder.

TA is short for Tech Animator in this case

MoCap is short for Motion Capture

And JC3 and JC4 both mean Just Cause 3 and Just Cause 4


JUST CAUSE 4

Overview

Gameplay Tools

Maya vs MoBu

What did we have?

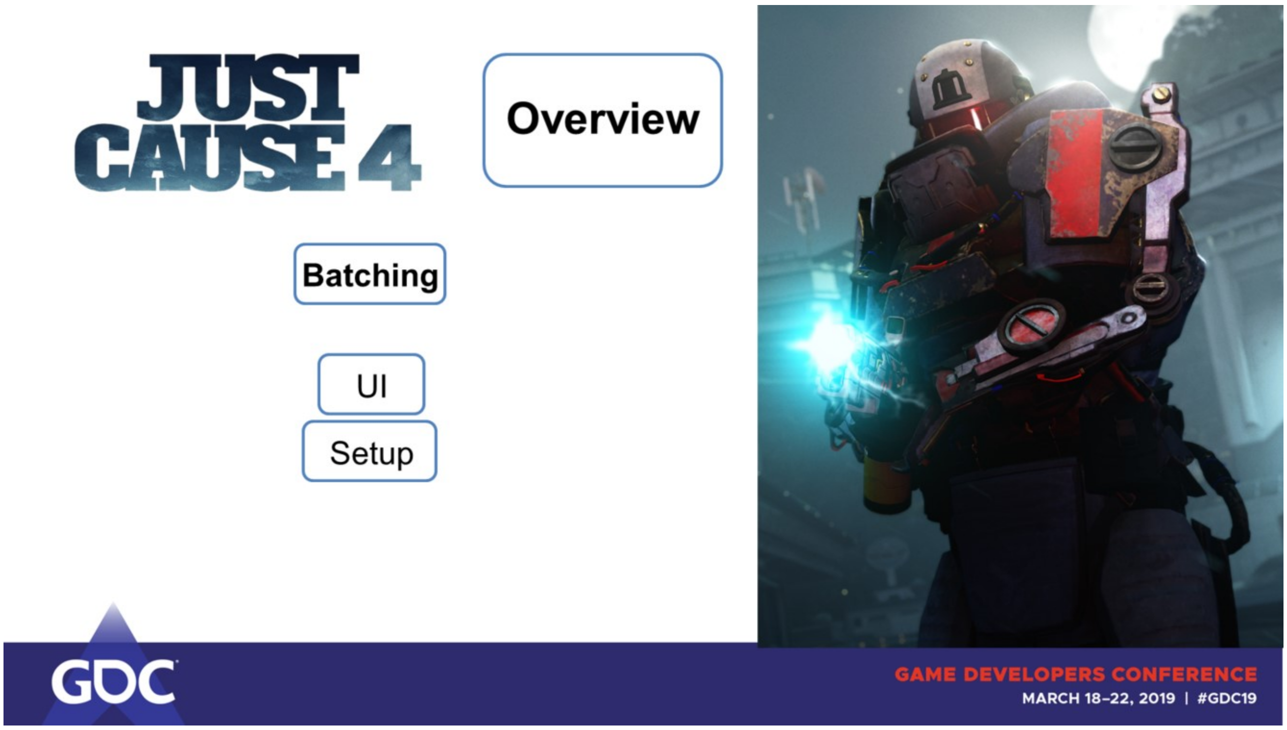
What did we change/add?

GDC

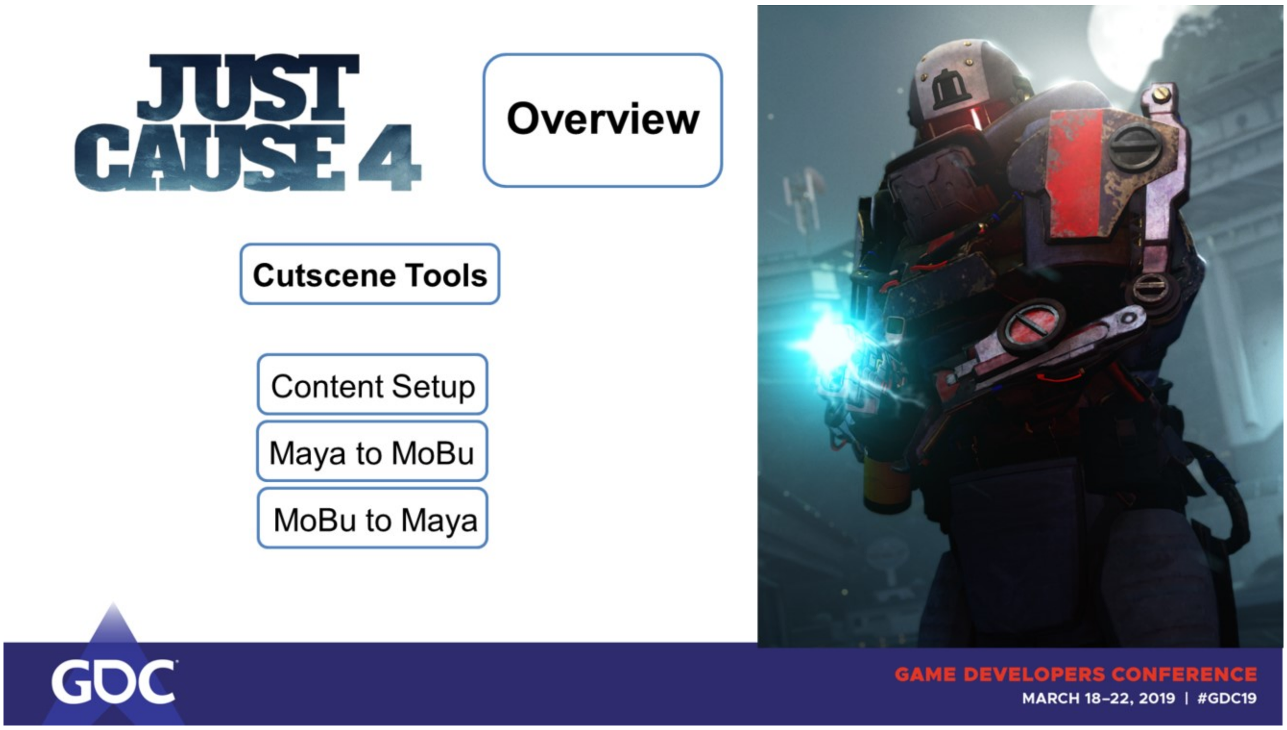
GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

Now that you know some key terms, here's brief overview of what's to come over the course of this talk.

I'm going to examine the 2 main DCC packages that we used, Maya and MotionBuilder. This includes what we were working with from JC3 and then what we did differently for JC4, additions made to the pipeline, and tasks that we tried to automate as much as possible.



And speaking of automation, I think it's worth looking at the batching process that was set up at the studio. A standalone UI was created that allowed us to quickly set up customized batch scripts for any tool or process we had in Maya.



JUST CAUSE 4

Overview

Cutscene Tools

Content Setup

Maya to MoBu

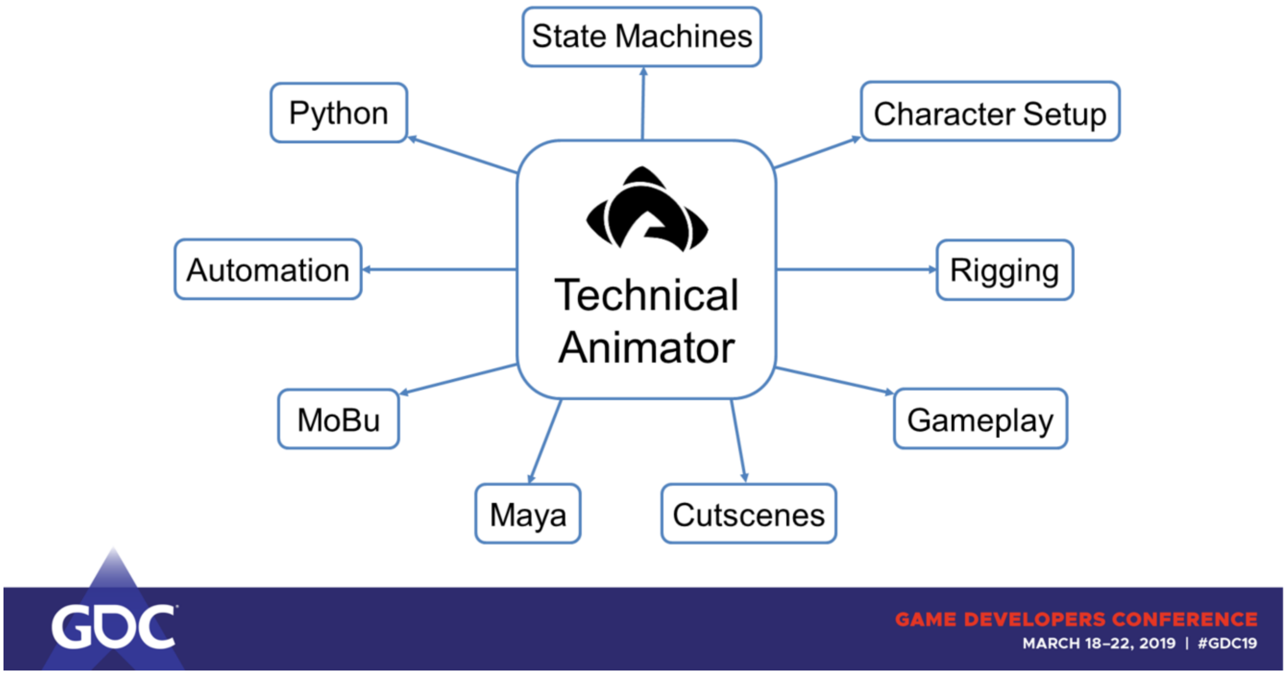
MoBu to Maya

GDC

GAME DEVELOPERS CONFERENCE

MARCH 18–22, 2019 | #GDC19

Finally, I'll end by going over our cutscene tools and how we set up the content for running the pipeline, as well as get a little in-depth on how the actual transferring of content between Maya and MotionBuilder was achieved.



In order to better understand a bit about my job at Avalanche studios, it's important to define the role of a technical animator at the Studio as every studio within the entire game development world seems to have a different definition for what a tech animator or even a tech artist should be. To illustrate this better I've thrown some buzz words up on the screen for you.

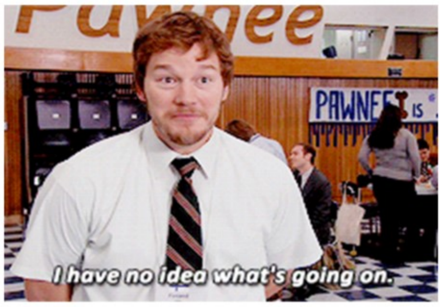
The Tech Animators at our studio all at the very least know a little bit of everything and have proven the ability to pick up new unknowns and very quickly adapt and learn. For myself alone, I've dug into everything involved with the animation process to some degree. I've built a ton of python based tools and scripts, supported both the gameplay and cutscene teams, dug into state machines and in-editor content, and have worked with each and every department in some way from the vehicle team, AI designers, graphics programmers…almost everyone.

Because of this, a TA at our studio needs to be ready to roll with the punches at any moment. They also need to solve the overall major problem that is "how will we maintain so much content, when we are limited in our time and personnel."

Another aspect of being a TA at Avalanche studios is that each project, such as Just Cause 4, theHunter: CoTW, or our upcoming titles Rage2 and Generation Zero, they all utilize a different workflow with DCC specific preferences. This leads me to the question for JC4 specifically…

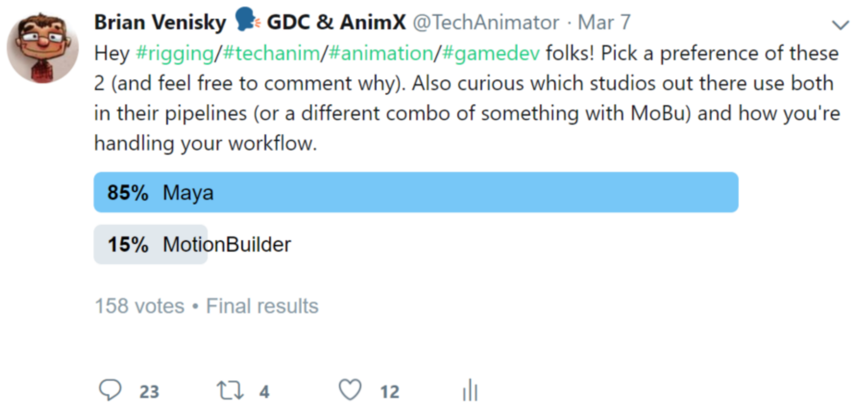
MAYA

MOBU



Maya or MotionBuilder?

While we were mainly using Maya for Just Cause 3, we did have some tools created during development that allowed for animators to push an animation from Maya into MoBu and also take an animation from MoBu and bring that back into Maya, however this was not extensive enough to have MoBu be anything other than a supplemental tool as opposed to a standalone. As a Maya guy, it did take me a while to actually appreciate what MotionBuilder can do for an animator, but I do now realize that the things it does well compared to Maya it does REALLY well. I've also seen first hand some passionate debates between folks on which one is the superior product. You'd be amazed at how heated they get.



Brian Venisky 🗣 GDC & AnimX @TechAnimator · Mar 7

Hey #rigging/#techanim/#animation/#gamedev folks! Pick a preference of these 2 (and feel free to comment why). Also curious which studios out there use both in their pipelines (or a different combo of something with MoBu) and how you're handling your workflow.

85% Maya

15% MotionBuilder

158 votes • Final results

23 4 12

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

I actually made a twitter poll recently while prepping for this talk that showed heavy Maya favoritism…however MoBu always has a passionate fan base and if you go back and check out this thread there ARE some great points made in favor of both DCCs…

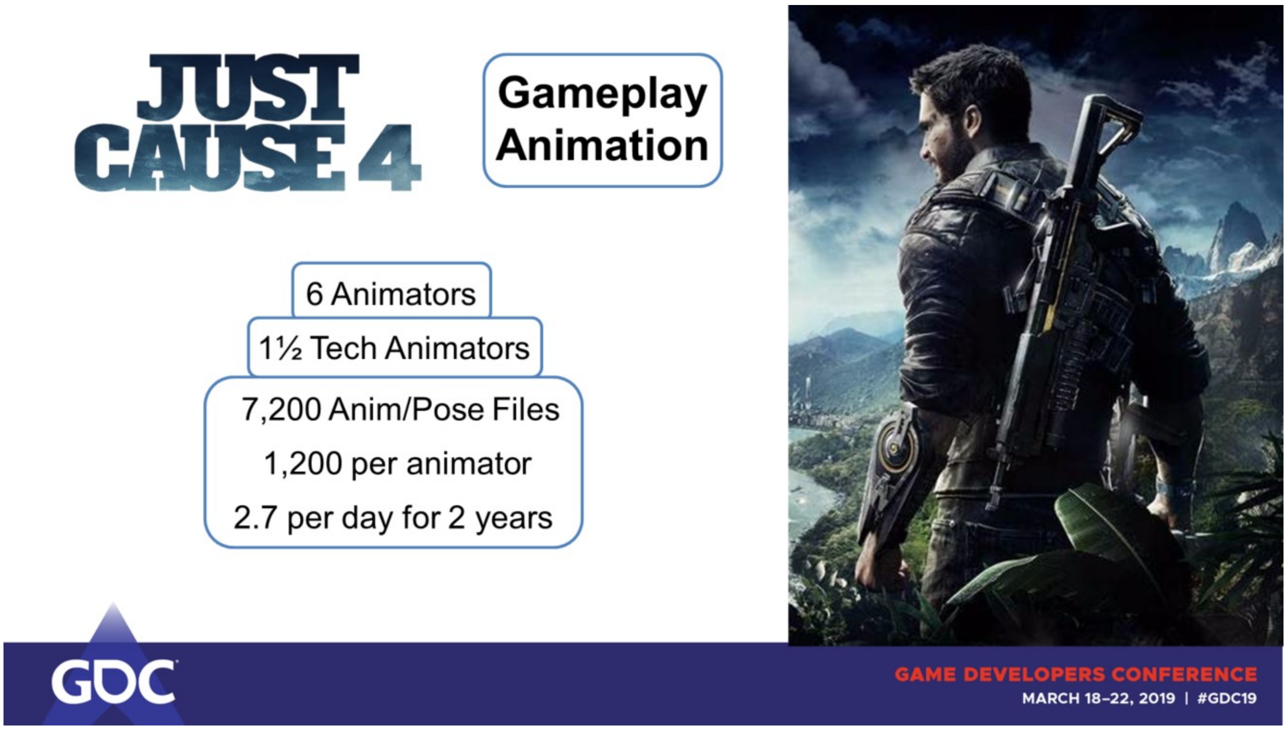
SO… with that in mind.

When asked Maya or MotionBuilder…

Why not both?

It almost seemed like a death sentence to say YES to building up a pipeline that could truly support two DCCs, considering at the time, there was only 1 tech animator, myself, and 6 animators to support in 2 pretty different pieces of software, BUT our team was split up on preference and expertise enough that I decided it was more beneficial to allow them all to work the way they wanted to and were comfortable with considering the lack of time we actually had to complete the project, especially since we were dealing with so many different factors that worked better in one or the other.

Since many other studios out there do in fact use both MotionBuilder and Maya, we knew that building up this pipeline wasn't a crazy idea, but the challenge would be to develop something that was robust enough with our lack of people-power.


JUST CAUSE 4

Gameplay Animation

6 Animators

1½ Tech Animators

7,200 Anim/Pose Files

1,200 per animator

2.7 per day for 2 years

GDC

GAME DEVELOPERS CONFERENCE

MARCH 18–22, 2019 | #GDC19

In order to try and explain how overloaded the team actually was in working on a AAA open-world title such as JC4 I want to start off with some quick facts.

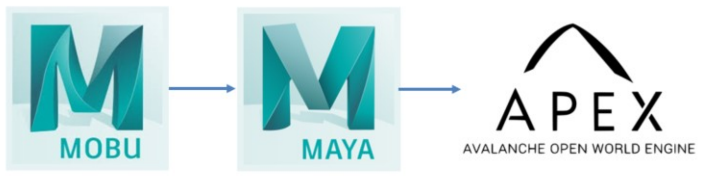
On the gameplay side of animation, we had 6 animators to support. And no this isn't a King Solomon cut the baby in half situation where we employed a half of a tech animator…1 and a half tech animators is my way to quantify being the only Tech Animator for the first 2 years of the of production before I had any sort of extra TA help later on.

In the end we had around 7,200 unique animations and amongst 6 animators that's quite daunting to manage. It equates to about 1,200 animations per animator for the whole project, very roughly 2.7 animations per working day over the course of 2 years (which is a rough estimate of the actual production time). Factor in the amount of time necessary for polish, things like xsens capturing, throwaway R&D work, and all those meetings that everyone loves to be part of day to day…aaand OOF

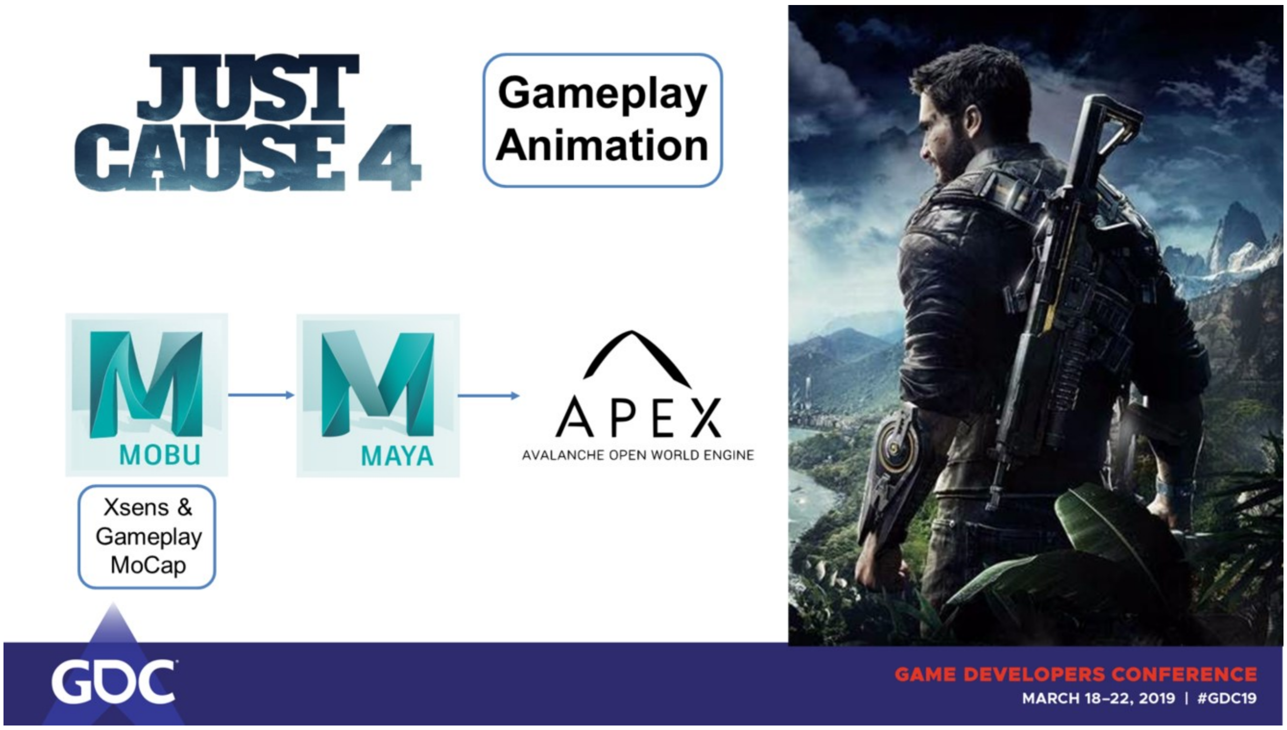
Now as far as preference for those 6 animators, we had 1 who was 100% motionbuilder and never touched maya, 1 who was 100% maya and never touched motiobuilder, and the other 4 had varying preference where they used both but each a bit differently.

JUST CAUSE 4

Gameplay Animation



But as far as pipeline goes, we did have to follow this structure, which many studios who utilize both Maya and MoBu tend to follow.



JUST CAUSE 4

Gameplay Animation

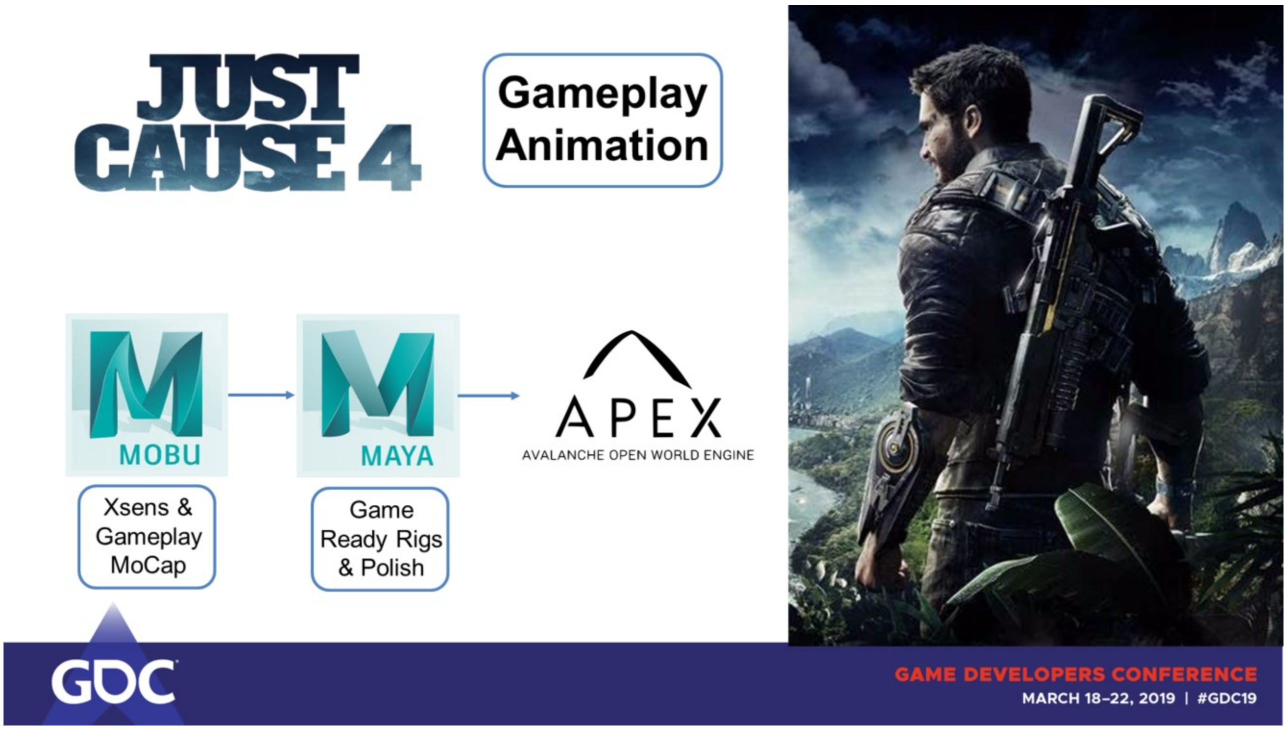
MOBU → MAYA → APEX AVALANCHE OPEN WORLD ENGINE

Xsens & Gameplay MoCap

GDC

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

You start in MoBu to handle any sort of MoCap data using the tools it has such as animation stitching, pinning, and the takes system for organizing content.



JUST CAUSE 4

Gameplay Animation

MOBU → MAYA → APEX AVALANCHE OPEN WORLD ENGINE

Xsens & Gameplay MoCap

Game Ready Rigs & Polish

GDC GAME DEVELOPERS CONFERENCE MARCH 18–22, 2019 | #GDC19

You then move that content over to Maya for animation polish and finaling, and then from Maya you export to your game.

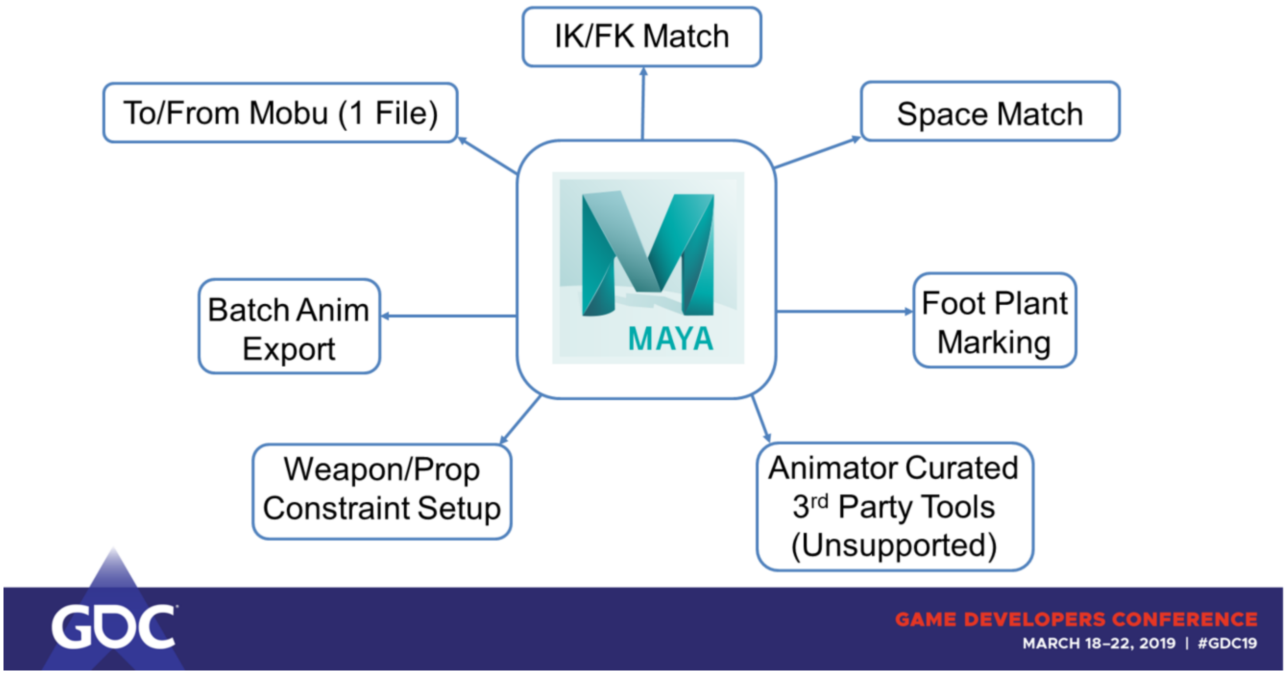
In our case, since we did have an animator that only used MoBu, he would stay in MoBu until he was ready to export to the game, and from there I had a process set up that would allow him to pass his content through Maya when exporting to the game because we did need to make sure that all of our correctives and custom rig features got picked up. Keep in mind that we were not using any sort of run-time animation, so it was extra important to stream everything through Maya on export to ensure quality control.

MAYA

GDC

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

Let’s dig into Maya now.


IK/FK Match
To/From Mobu (1 File)
Space Match
MAYA
Batch Anim Export
Foot Plant Marking
Weapon/Prop Constraint Setup
Animator Curated 3rd Party Tools (Unsupported)
GDC
GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

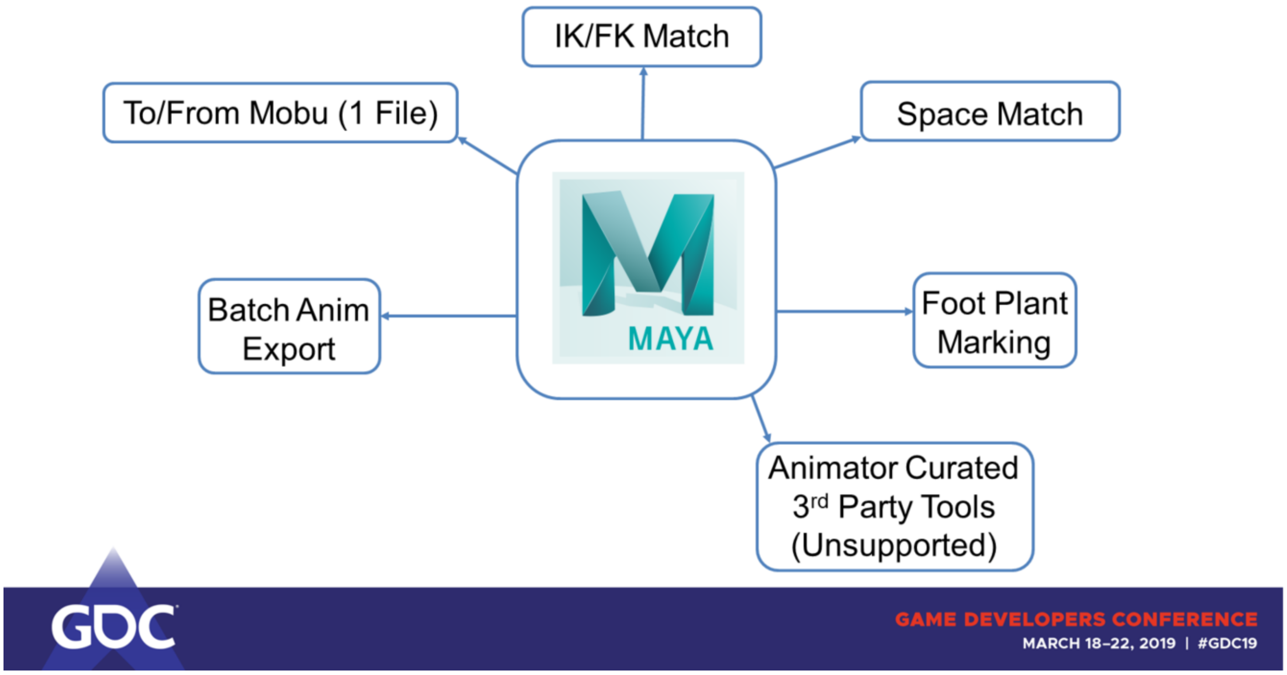
I needed to think about what we already had from JC3 and what we needed to do for JC4 that would get us where we needed to go. We already had many basic tools you'd probably expect to have in a AAA game pipeline, but after working with them for an entire project cycle, we were able to pinpoint the pros and cons, figuring out what to improve on.

Matching parent spaces and ik/fk limbs, as well as setting up constraints have been very standard for years, but how could we improve them or even use them differently than before? Did the tools we had overcomplicate things?

In the case of foot planting, animators on JC3 had to manually tag in the content when this was occurring. Could we automate that?

How about all those scripts and random tools that animators find online, install or use, and then they end up breaking Maya. Was there a better way to handle that and essentially "animator-proof" things a bit?

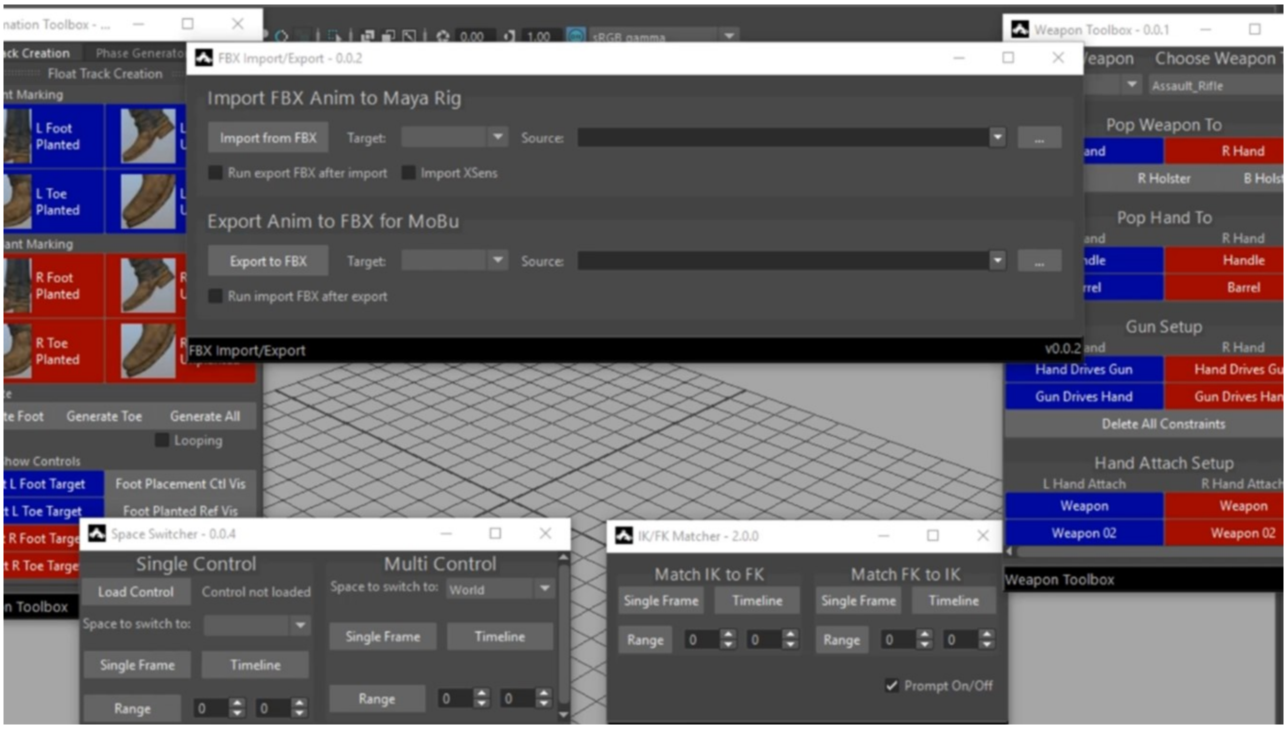
There were attachment tools that would help set up weapons and props for animators, but they ended up being so cumbersome with too many buttons and options that we actually…


IK/FK Match
To/From Mobu (1 File)
Space Match
MAYA
Batch Anim Export
Foot Plant Marking
Animator Curated 3rd Party Tools (Unsupported)
GDC
GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

…just got rid of them. Animators tended to have their own workflows set up to set up attachments and a lot of this was part of those outside tools they found so I already didn't have to worry about that.

We also had a file browser for the animators where they could find and open Maya files from within Maya as well as multi-select files and folders for batch exporting animation to the game.

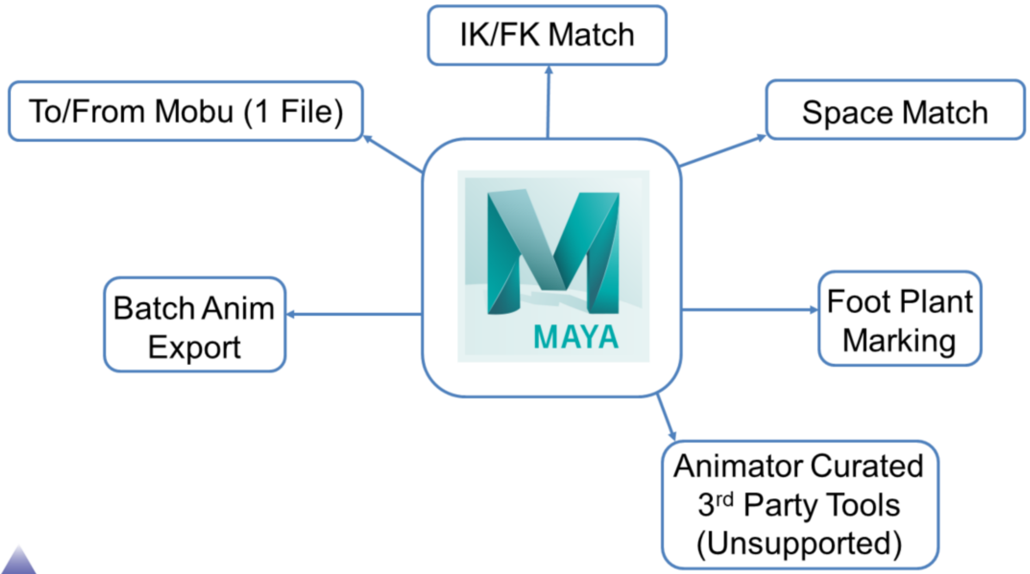
Finally, a tool to go from Maya to MotionBuilder and back to Maya existed, but it was very limited and we didn't have a good system to transfer multiple takes back into Maya or multiple files from Maya over to MoBu quickly.



Actually visualizing some of this now…

We had a lot of those processes broken down into 1 button clicks thinking it would be easy and awesome for hotkeying, but in many places this wasn’t very efficient. This did allow for a few things like space and ik/fk matching to have the ability to be used as hotkeys, but the physical UI was sort of a nightmare and a hassle to navigate. All of our tools also had their own dedicated UI window which was a pain to keep track of as well as organize visually.

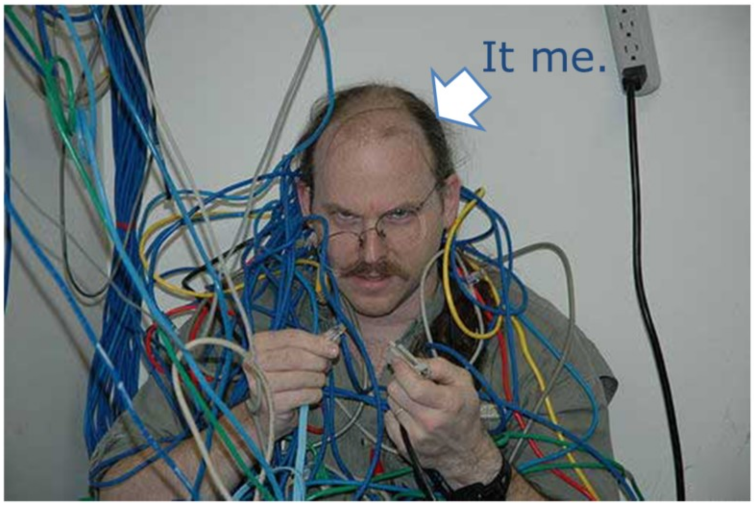
In addition to these tools it was sort of the wild west when animators started to download those 3rd party tools and install them themselves or create random shelf buttons for mel and python scripts.



Going back a slide, we did have animator organized 3rd party tools, but they were completely unsupported by tech animation. Essentially we gave authority to 1 animator who oversaw this process. He used a network folder to add scripts to and then called them in a custom userSetup file that any animator could grab.

It worked well enough but it wasn't something I could easily oversee myself and quality check.

Because of this, all of our supported as well as unsupported tools…



…Came together like that shoebox full of cables and wall chargers that you keep in the closet "just in case" you need them someday.

For JC4 instead of trying to add more, the focus was to see where we could scale back. Like those constraint tools, what else didn't we need? What needed a restructure? What tools needed serious code refactoring?

When it came down to it, we almost started from scratch on many things. There was also a lot of refactoring and some decisions made to either completely get rid of things that were barely if ever used, as well as a push to automate as many things as possible.

MORE ANIMATOR POWER!!

GDC GAME DEVELOPERS CONFERENCE MARCH 18–22, 2019 | #GDC19

This led us to a main goal. Instead of trying to solve every little issue… we would give more power to the animators!

We realized that we didn't have to try and make every little thing a button or try to solve every single problem, and we also realized that the animators in the end didn't want to work that way. Simplicity became the goal.

GOAL: Empower the Animators!

Efficiency

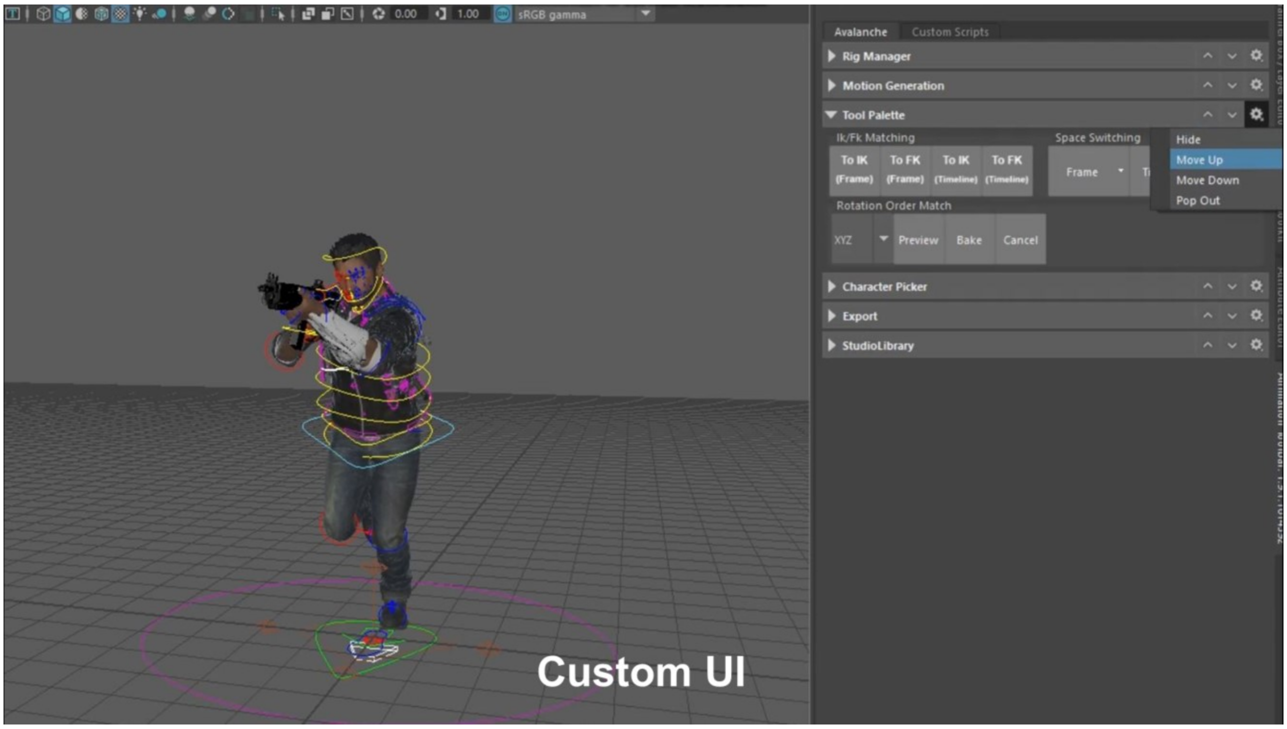
Speed

User Preference

Customization

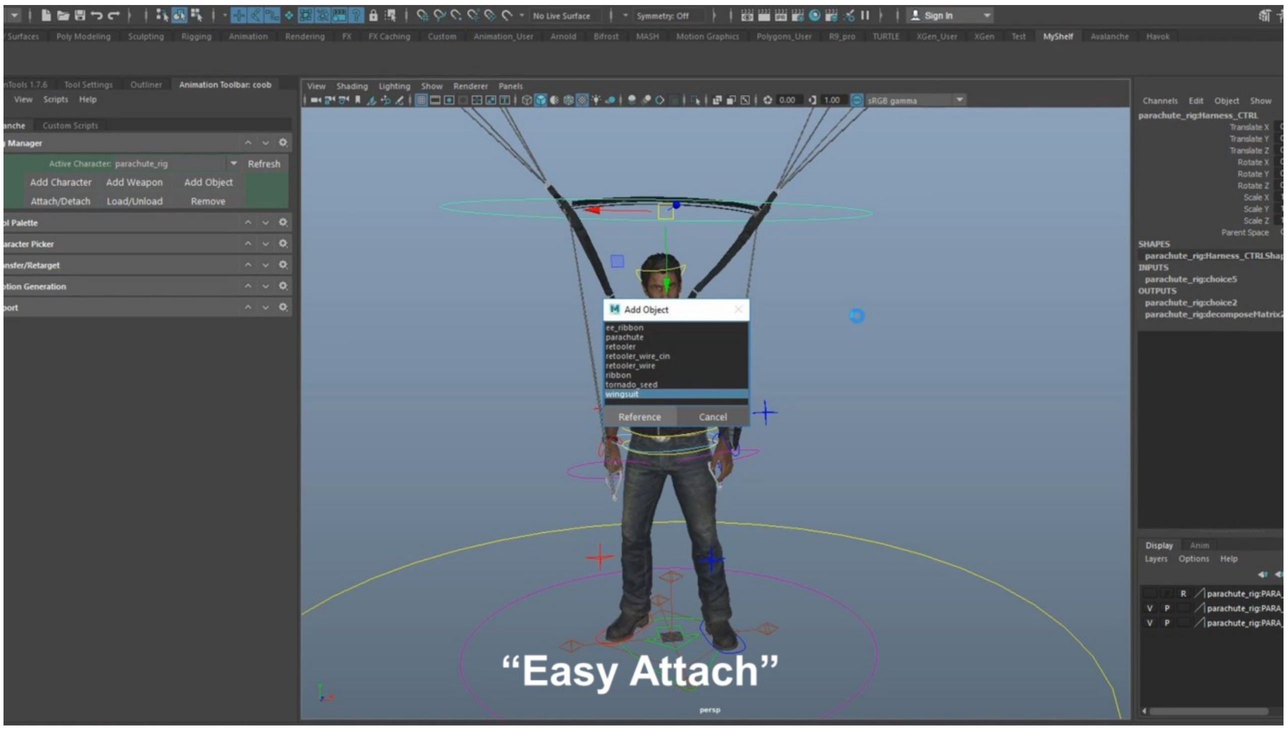
MAYA

In order to reach our goal of simplicity we started focusing on efficiency, speed, user preference, and customization.

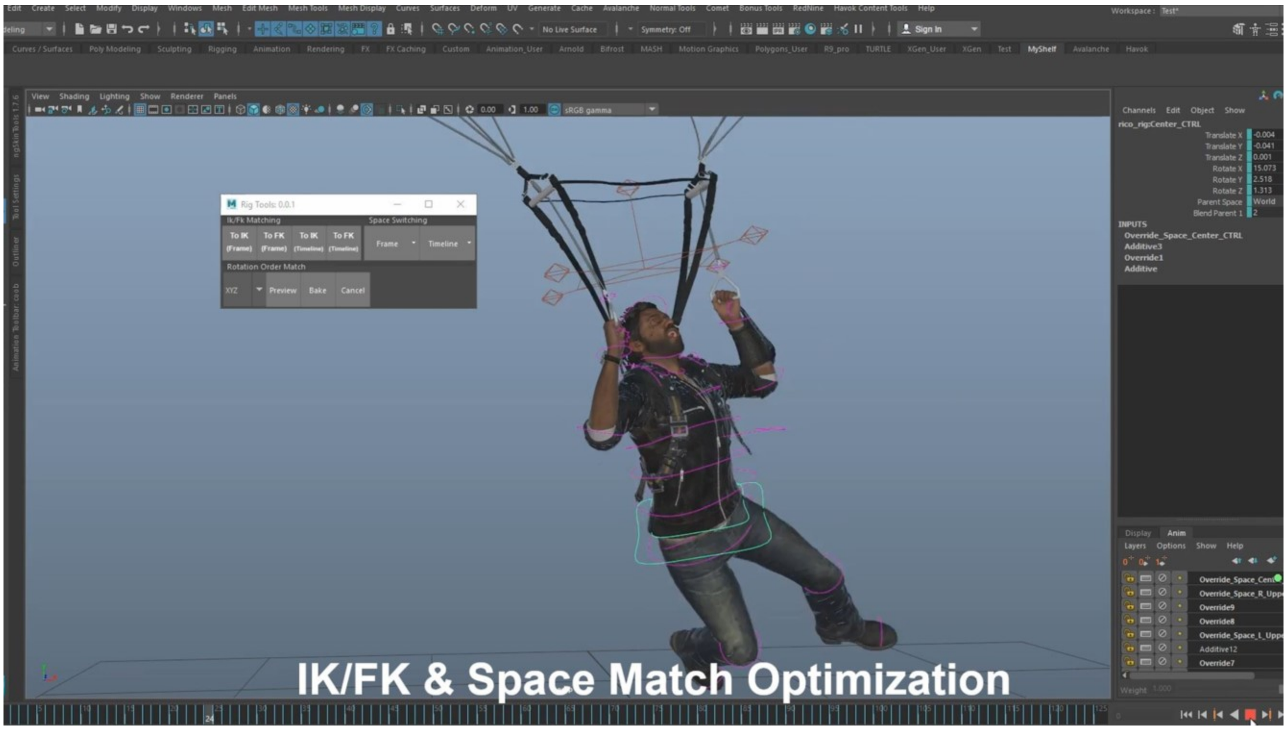


The first thing we did was tie all of our tools into one main toolbar that can be set and customized as the animators themselves want. After watching other GDC talks or checking out online resources, I realized that many other studios tie their tools into one animation specific toolbar, so I wanted to set out to first do that…but also give it a bit of something extra.

If the animation tools are open on Maya close, the tools open when you reopen Maya. They can be docked or undocked. You can pop tools out if you only need one but don’t want the entire UI in your way. If you like the UI but don’t use certain tools you can completely hide them from the toolset. You can also rearrange the order of the tools if you use one more than others and prefer it at the top of the stack.

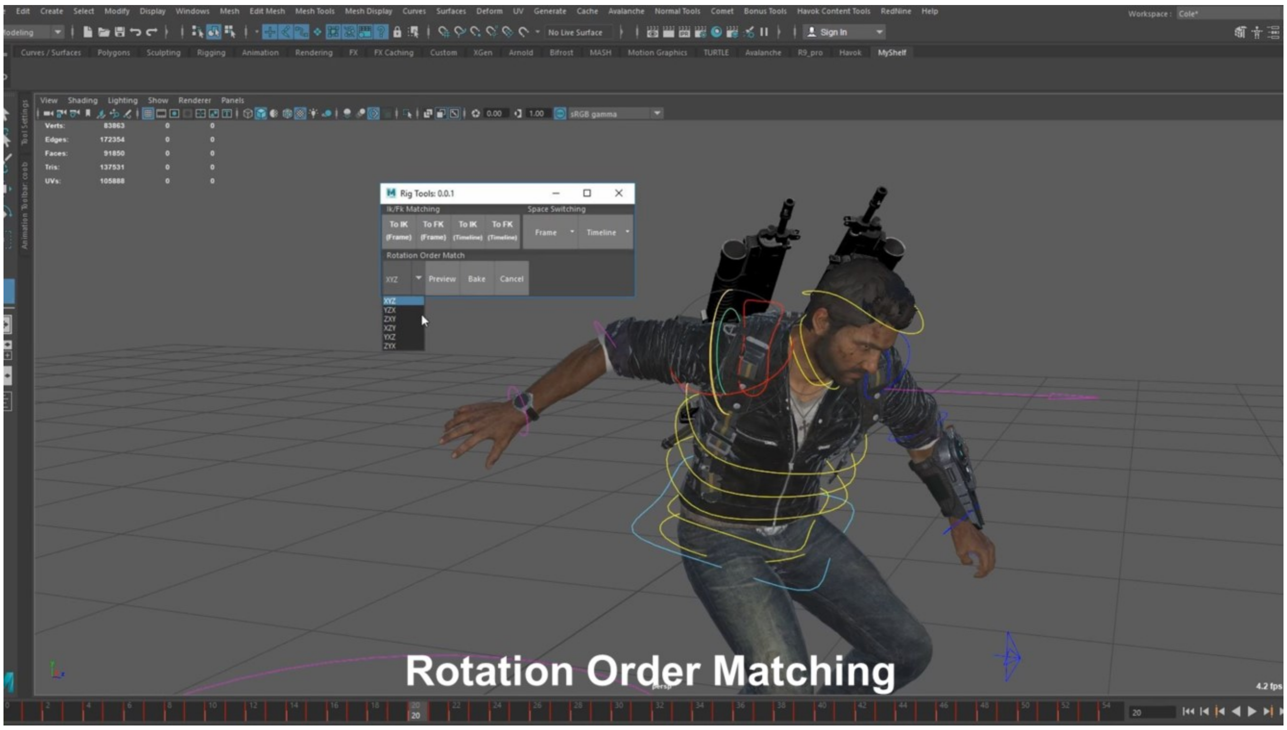


Instead of having animators click a ton of buttons or use convoluted interfaces to set attachments, for JC4 we created a simple 1 button attach/detach system that could easily setup specific attachments as you can see here with the parachute and wingsuit. Underneath the hood this process detects which rig is active, and when the attach function is run, sets up the active rig accordingly. Further, thanks to the power of space switching, once attached you could quickly set specifics as to how you wanted things like handles or buckles to follow the character.

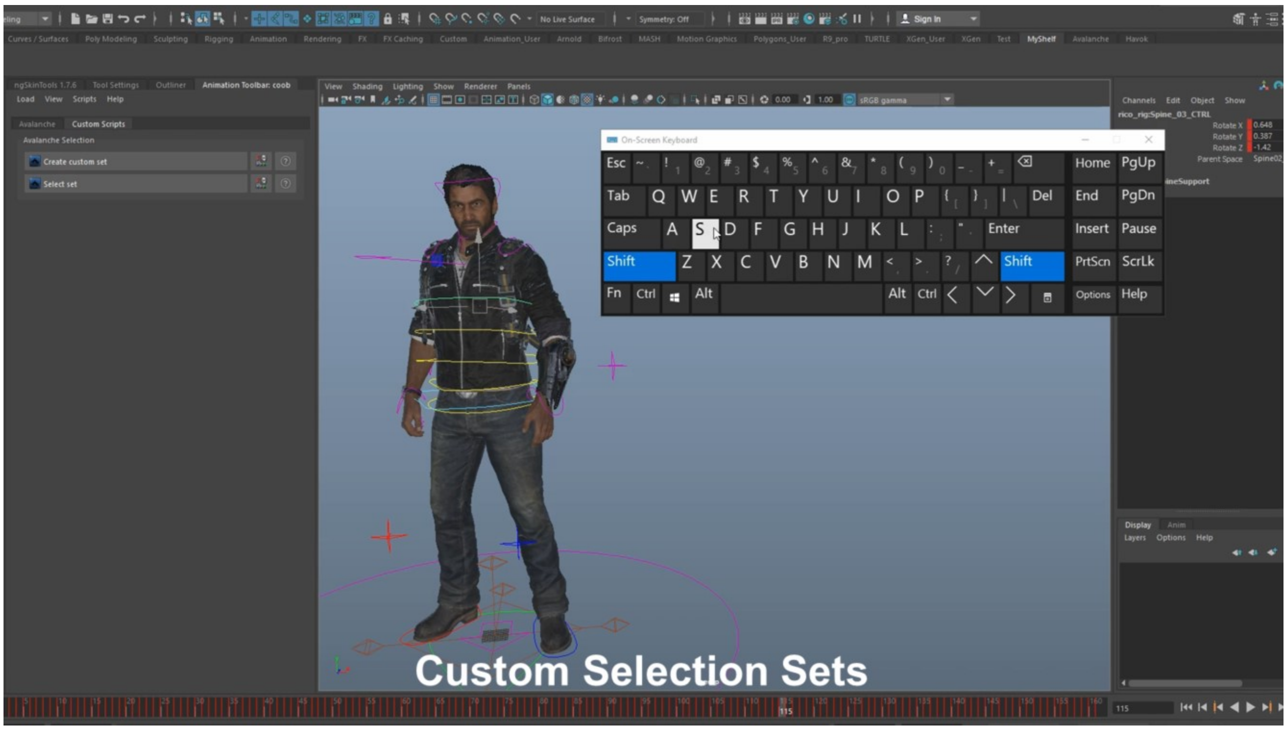


IK/Fk and Space matching are pretty standard these days, so what we set out to do instead of reinventing all this, was to evaluate and refactor the way we ran these tools to create quicker times for both running Ik/Fk matching and Space Switching. Single frame matching now happened instantaneously and if you wanted to match for the entire timeline, it only took a matter of seconds.

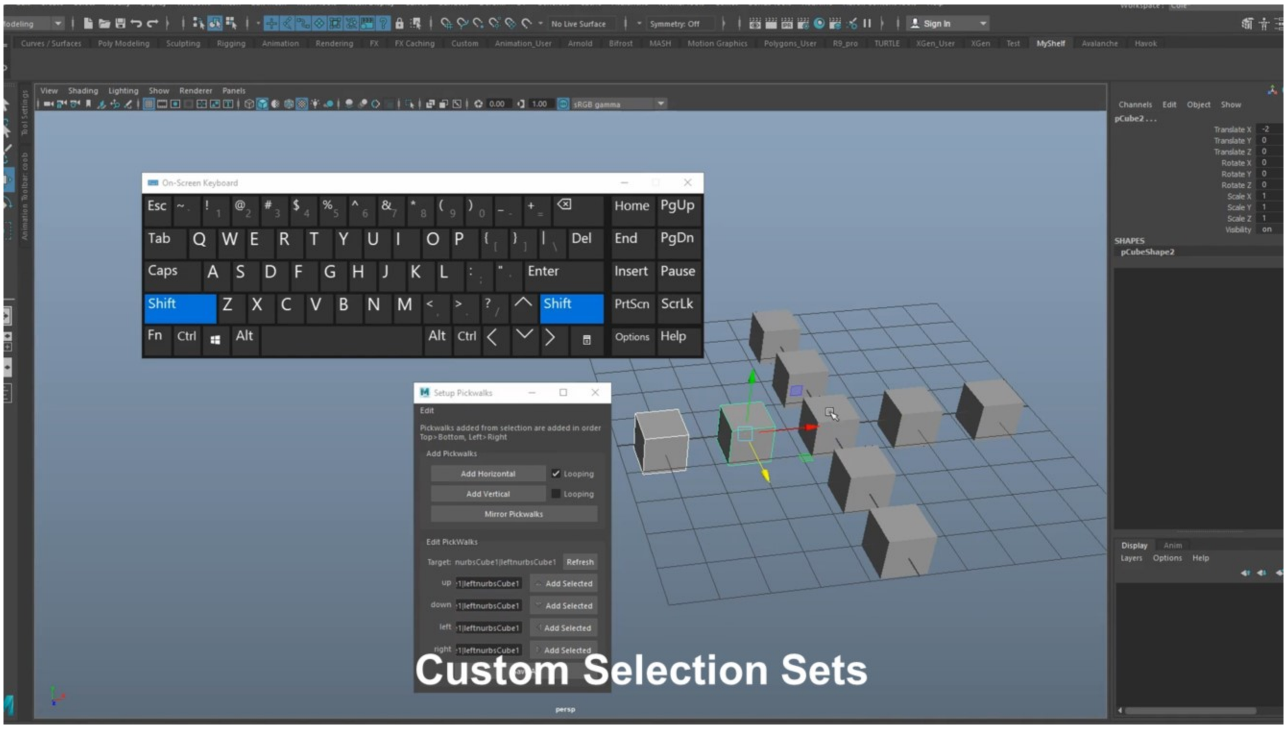
Something that was introduced back on JC3 to our rigs and was further enhanced by the quicker speeds of space matching was what we called the “driver root” control. You can see it here being that giant control above the character’s head that is controlling the entire character. Say you want to animate the character here dangling, with both arms and legs in IK. You can easily place the driver root to the position you want to rotate from, then set the space for both hands, feet, and the COG control to this driver root, animate it tilting left and right, and then space match all of these controls at once back to world space when you can then polish each one individually. I like to call this control a “locator on steroids” as it functions the same way many animators use locators to bake and transfer animation, but having it built into the rig and ready to work with space matching gives you immediate out of the box functionality the second you reference a character in your scene.



We also introduced rotation order matching. The rigs themselves were set up with best practice rotate orders on all controls, but adding the ability for an animator to change the rotate orders of controls on the fly gave them even more control.



We also gave the animators the ability to create custom selection sets on the fly that were also simple to make hot-keyable. Multi-Select the controls you'd like to group together, click "create selection set" and that's it. Now you've got a custom group of controls you can grab instantly.



Edit

Pickwalks added from selection are added in order Top>Bottom, Left>Right

Add Pickwalks

Add Horizontal ✓ Looping

Add Vertical Looping

Mirror Pickwalks

Edit PickWalks

Target: nurbsCube1|leftnurbsCube1 Refresh

up -1|leftnurbsCube1 Add Selected

down -1|leftnurbsCube1 Add Selected

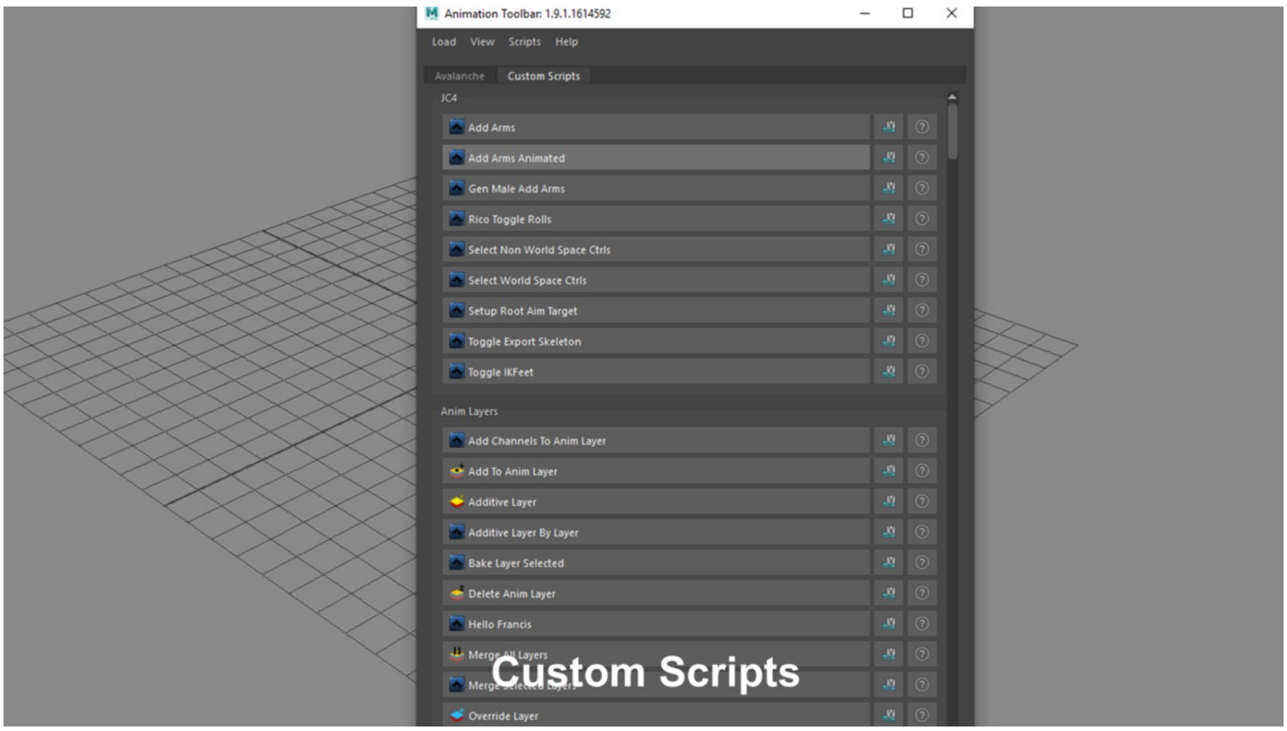
left -1|leftnurbsCube1 Add Selected

right -1|leftnurbsCube1 Add Selected

Even more power was granted to the animators to be able set up their own custom pickwalking for ANYTHING.

On JC3 we had pickwalking on the character rigs only, and in order to set those settings, we would have to hard code it into the rig.

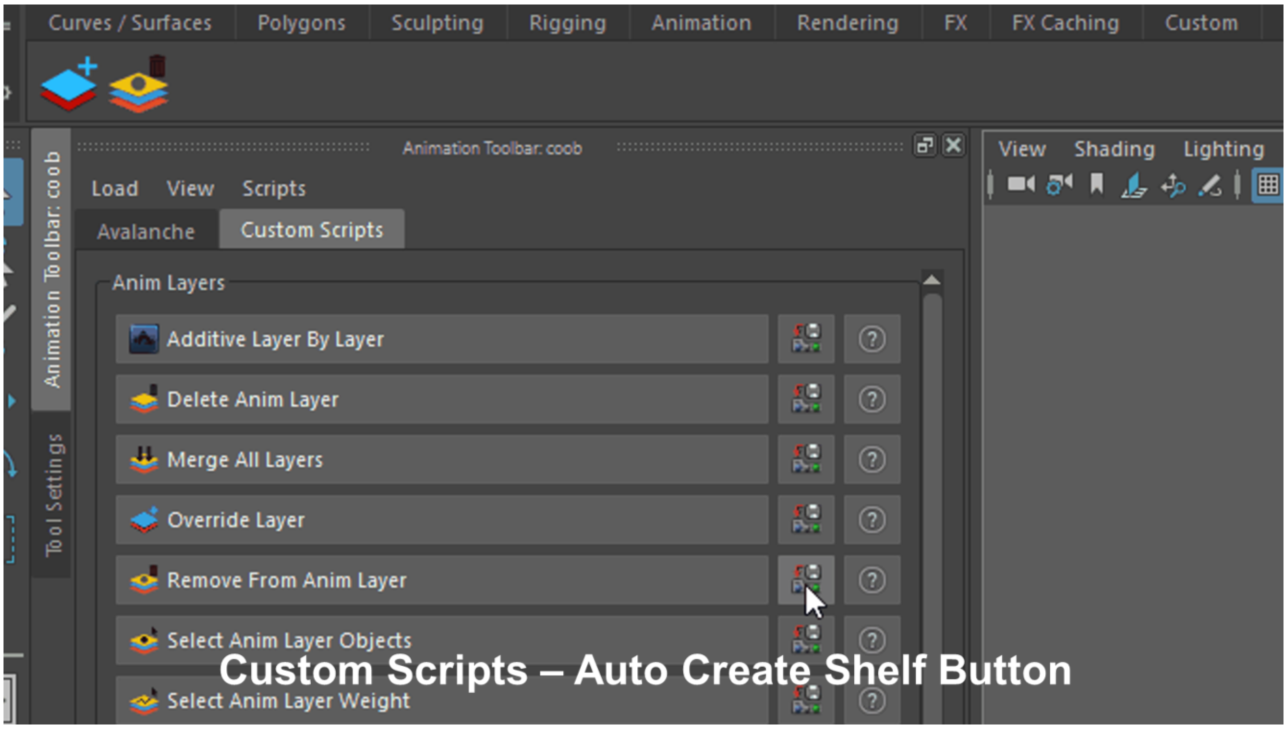
This time around, the character rigs DID have default pickwalk settings, but animators could customize this any way they'd like as well, and also add pickwalking to whatever else they may choose to.



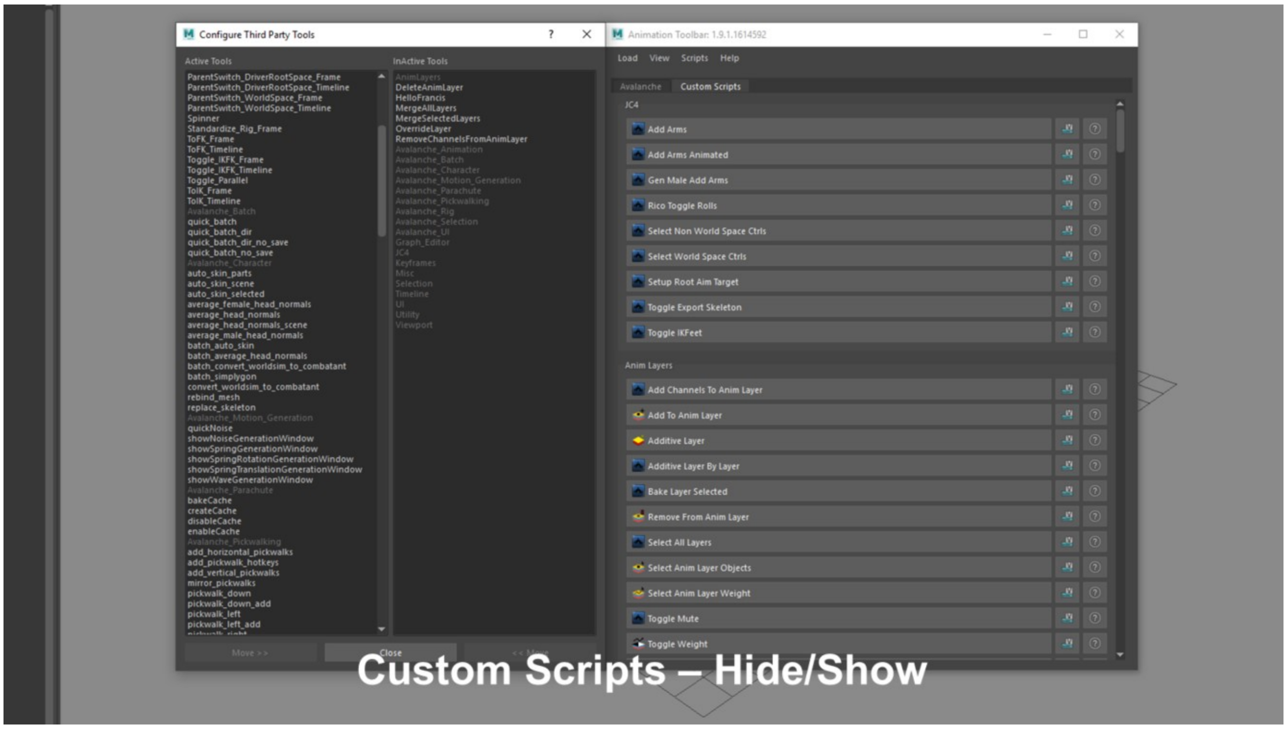
We also added a custom scripts tab to the toolset. That tangled mess of 3rd party scripts and snippets were turned into “official, unofficial scripts”.

The TAs can add 3rd party tools officially in perforce for deployment and animators can locally add anything they want to their Maya scripts folder and then magically populate that tool nice and neatly here. There’s no need to override or create a custom userSetup file, it’s all here for you to use pretty much out of the box.

All those random MEL and python scripts that were now nice and neatly organized and labeled in a way that anyone can easily see and understand.



All custom scripts have an auto-create shelf button.



Configure Third Party Tools

Active Tools

```
ParentSwitch_DriverRootSpace_Frame
ParentSwitch_DriverRootSpace_Timeline
ParentSwitch_WorldSpace_Frame
ParentSwitch_WorldSpace_Timeline
Spinner
Standardize_Rig_Frame
ToFK_Frame
ToFK_Timeline
Toggle_IKFK_Frame
Toggle_IKFK_Timeline
Toggle_Parallel
ToIK_Frame
ToIK_Timeline
Avalanche_Batch
quick_batch
quick_batch_dir
quick_batch_dir_no_save
quick_batch_no_save
Avalanche_Character
auto_skin_parts
auto_skin_scene
auto_skin_selected
average_female_head_normals
average_head_normals
average_head_normals_scene
average_male_head_normals
batch_auto_skin
batch_average_head_normals
batch_convert_worldsim_to_combatant
batch_simplygon
convert_worldsim_to_combatant
rebind_mesh
replace_skeleton
Avalanche_Motion_Generation
quickNoise
showNoiseGenerationWindow
showSpringGenerationWindow
showSpringRotationGenerationWindow
showSpringTranslationGenerationWindow
showWaveGenerationWindow
Avalanche_Parachute
bakeCache
createCache
disableCache
enableCache
Avalanche_Pickwalking
add_horizontal_pickwalks
add_pickwalk_hotkeys
add_vertical_pickwalks
mirror_pickwalks
pickwalk_down
pickwalk_down_add
pickwalk_left
pickwalk_left_add
```

InActive Tools

```
AnimLayers
DeleteAnimLayer
HelloFrancis
MergeAllLayers
MergeSelectedLayers
OverrideLayer
RemoveChannelsFromAnimLayer
Avalanche_Animation
Avalanche_Batch
Avalanche_Character
Avalanche_Motion_Generation
Avalanche_Parachute
Avalanche_Pickwalking
Avalanche_Rig
Avalanche_Selection
Avalanche_UI
Graph_Editor
JC4
Keyframes
Misc
Selection
Timeline
UI
Utility
Viewport
```

Animation Toolbar: 1.9.1.1614592

JC4

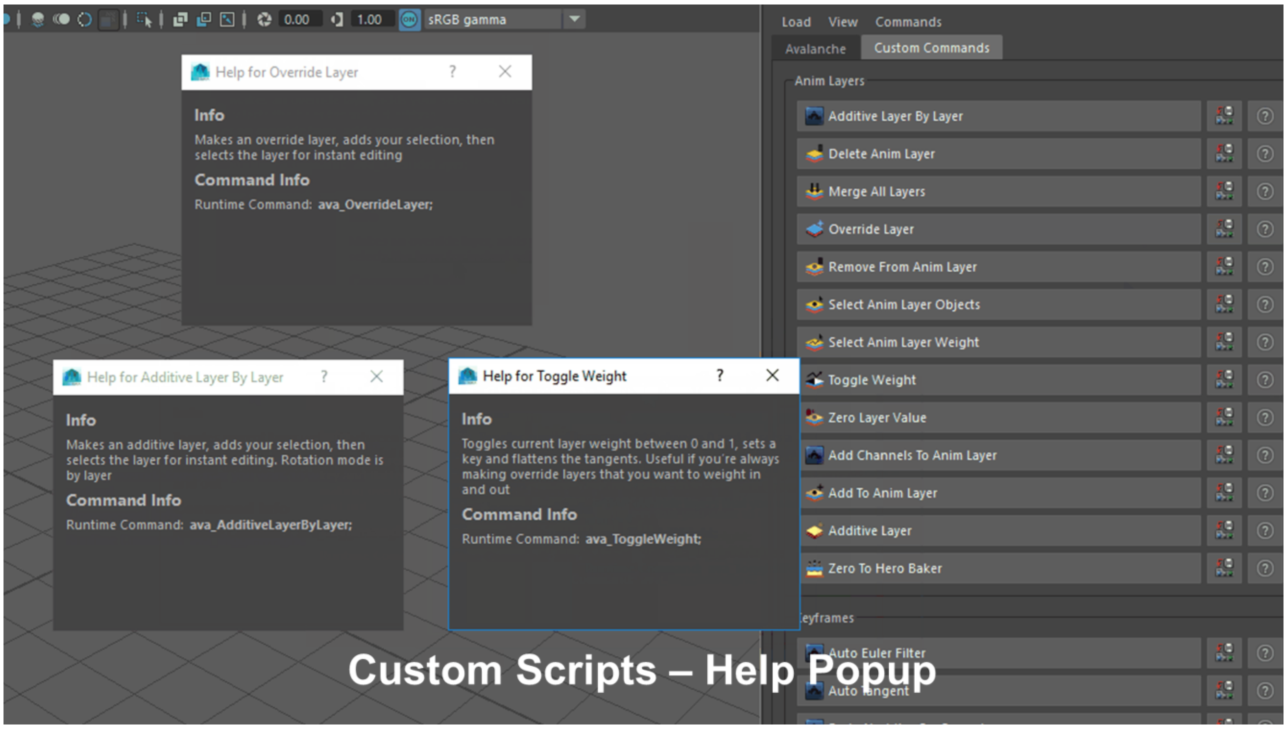
- Add Arms
- Add Arms Animated
- Gen Male Add Arms
- Rico Toggle Rolls
- Select Non World Space Ctrls
- Select World Space Ctrls
- Setup Root Aim Target
- Toggle Export Skeleton
- Toggle IKFeet

Anim Layers

- Add Channels To Anim Layer
- Add To Anim Layer
- Additive Layer
- Additive Layer By Layer
- Bake Layer Selected
- Remove From Anim Layer
- Select All Layers
- Select Anim Layer Objects
- Select Anim Layer Weight
- Toggle Mute
- Toggle Weight

Custom Scripts – Hide/Show

The ability to hide/show what you want like our official tools.



Help for Override Layer

Info

Makes an override layer, adds your selection, then selects the layer for instant editing

Command Info

Runtime Command: ava_OverrideLayer;

Help for Additive Layer By Layer

Info

Makes an additive layer, adds your selection, then selects the layer for instant editing. Rotation mode is by layer

Command Info

Runtime Command: ava_AdditiveLayerByLayer;

Help for Toggle Weight

Info

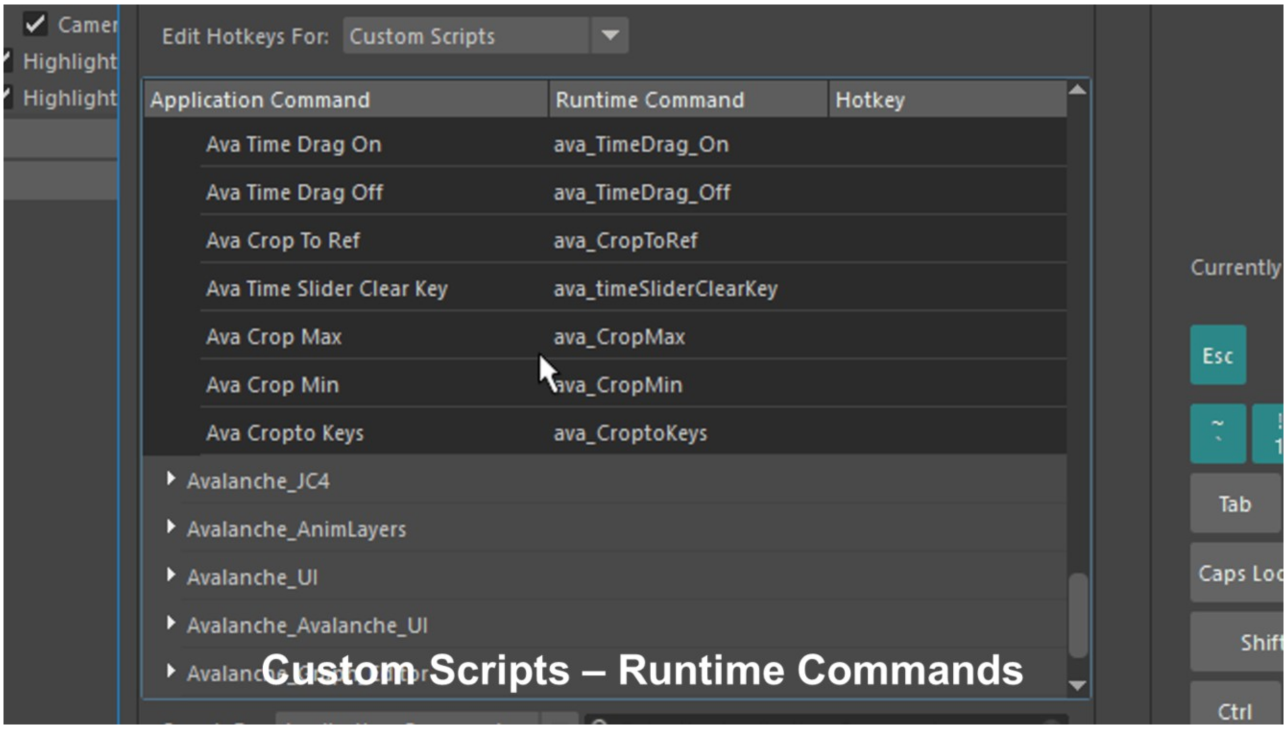
Toggles current layer weight between 0 and 1, sets a key and flattens the tangents. Useful if you're always making override layers that you want to weight in and out

Command Info

Runtime Command: ava_ToggleWeight;

Custom Scripts – Help Popup

And a customized help popup that includes a unique command.

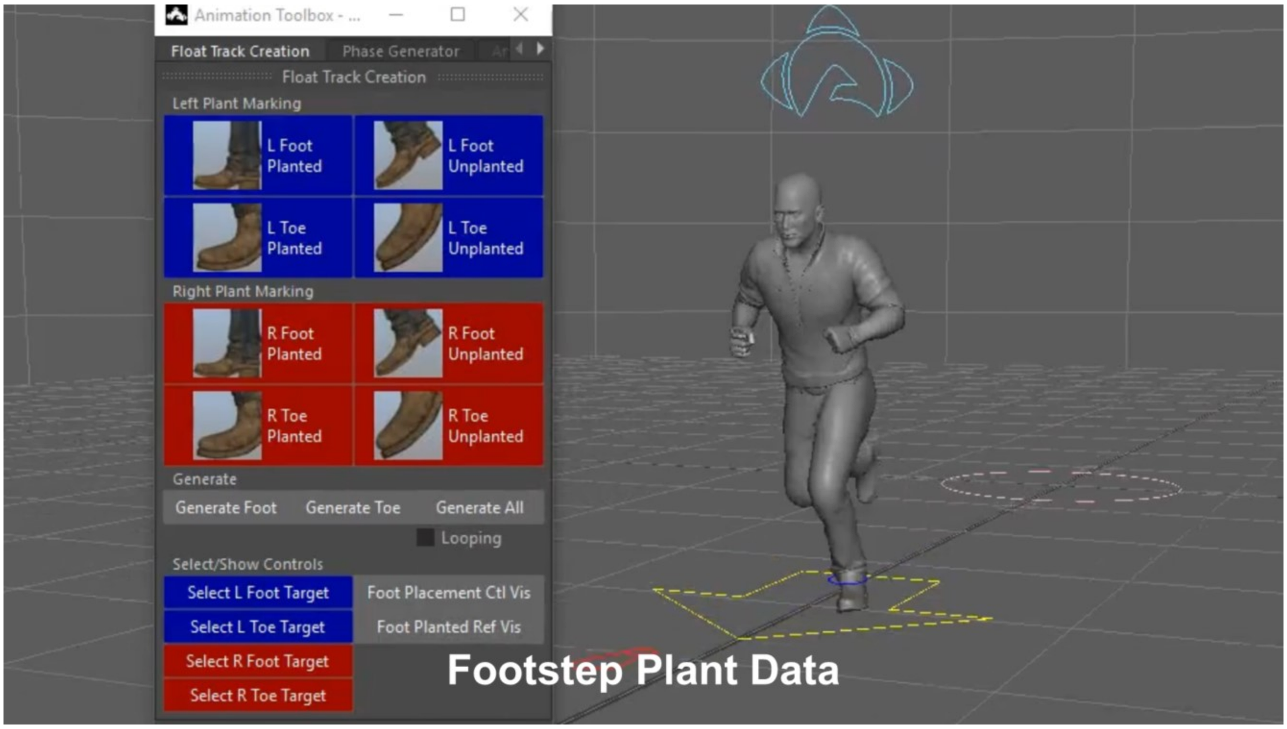


Edit Hotkeys For: Custom Scripts

Application Command	Runtime Command	Hotkey
Ava Time Drag On	ava_TimeDrag_On	
Ava Time Drag Off	ava_TimeDrag_Off	
Ava Crop To Ref	ava_CropToRef	
Ava Time Slider Clear Key	ava_timeSliderClearKey	
Ava Crop Max	ava_CropMax	
Ava Crop Min	ava_CropMin	
Ava Cropto Keys	ava_CroptoKeys	
▸ Avalanche_JC4		
▸ Avalanche_AnimLayers		
▸ Avalanche_UI		
▸ Avalanche_Avalanche_UI		

Custom Scripts – Runtime Commands

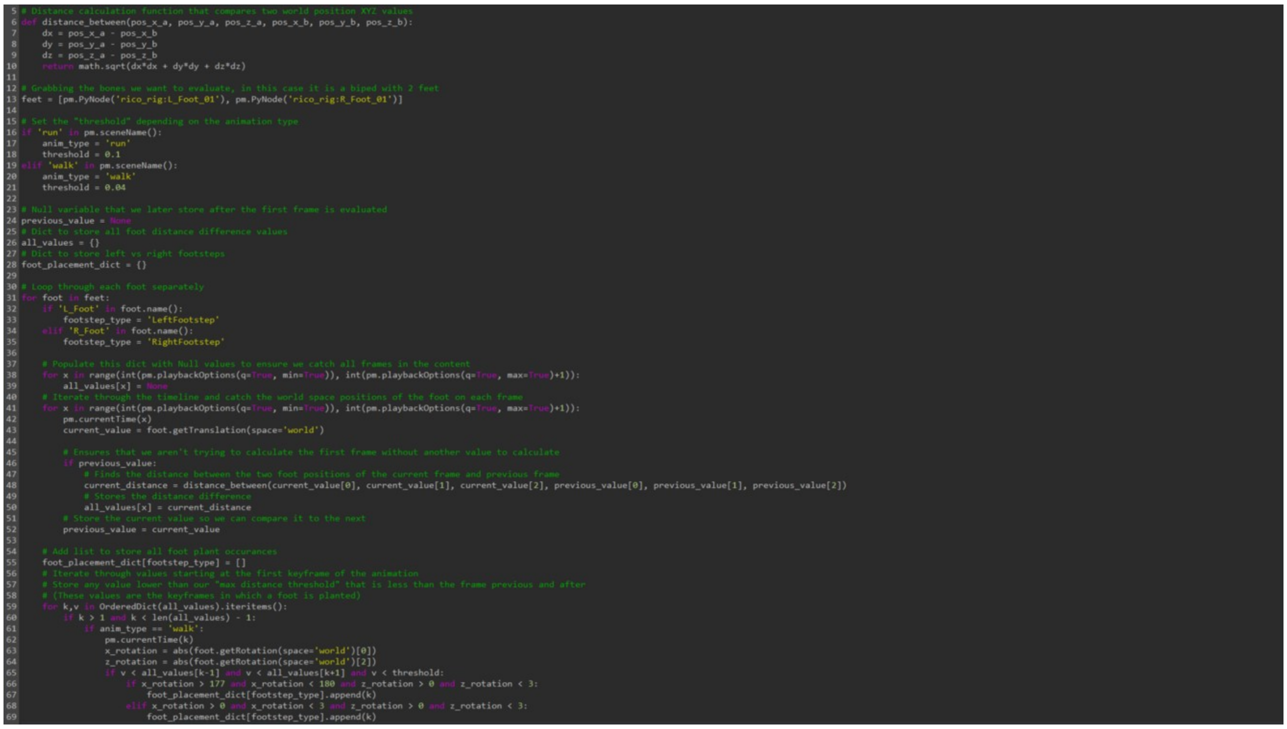
Which through use of metadata, create runtime commands when sourced to allow for the animators to easily apply to hotkeys and marking menus.



I want to show you one specific example of workflow improvements that actually took away power from the animators, but it was not power that they really wanted.

On JC3 anytime we wanted to grab footstep data from a file we needed to have an animator go and manually tag anytime a footstep occurred. They had a tool that allowed them to easily scrub through an animation and click left plant, right plant, left plant, etc as they did so. You can see here the blue and red foot placement markers that are getting set for all frames where a foot hits the ground. While this was a very simple and easy to do, it was monotonous took up lots of time. Time that took away from say, actual animation work being done.

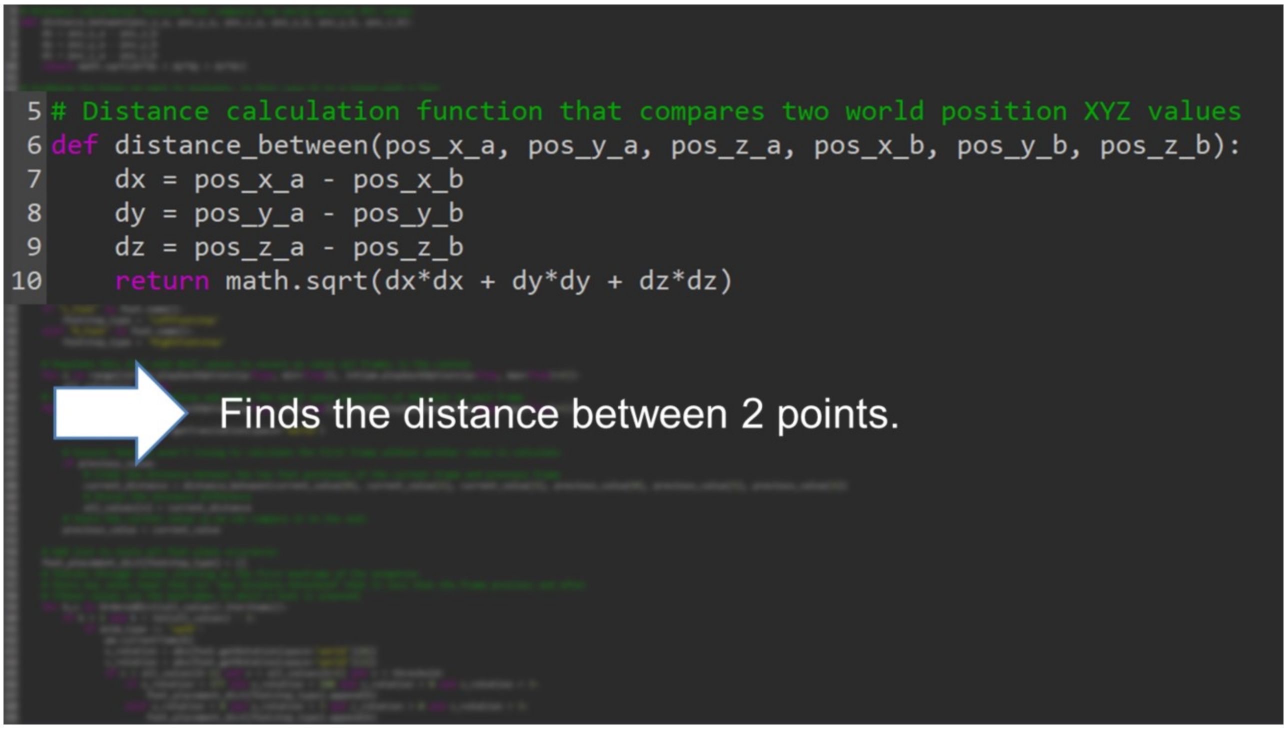
On JC4, I wanted to take this out of their hands and automate the process completely.



```python
# Distance calculation function that compares two world position XYZ values
def distance_between(pos_x_a, pos_y_a, pos_z_a, pos_x_b, pos_y_b, pos_z_b):
    dx = pos_x_a - pos_x_b
    dy = pos_y_a - pos_y_b
    dz = pos_z_a - pos_z_b
    return math.sqrt(dx*dx + dy*dy + dz*dz)

# Grabbing the bones we want to evaluate, in this case it is a biped with 2 feet
feet = [pm.PyNode('rico_rig:L_Foot_01'), pm.PyNode('rico_rig:R_Foot_01')]

# Set the "threshold" depending on the animation type
if 'run' in pm.sceneName():
    anim_type = 'run'
    threshold = 0.1
elif 'walk' in pm.sceneName():
    anim_type = 'walk'
    threshold = 0.04

# Null variable that we later store after the first frame is evaluated
previous_value = None
# Dict to store all foot distance difference values
all_values = {}
# Dict to store left vs right footsteps
foot_placement_dict = {}

# Loop through each foot separately
for foot in feet:
    if 'L_Foot' in foot.name():
        footstep_type = 'LeftFootstep'
    elif 'R_Foot' in foot.name():
        footstep_type = 'RightFootstep'

    # Populate this dict with Null values to ensure we catch all frames in the content
    for x in range(int(pm.playbackOptions(q=True, min=True)), int(pm.playbackOptions(q=True, max=True)+1)):
        all_values[x] = None
    # Iterate through the timeline and catch the world space positions of the foot on each frame
    for x in range(int(pm.playbackOptions(q=True, min=True)), int(pm.playbackOptions(q=True, max=True)+1)):
        pm.currentTime(x)
        current_value = foot.getTranslation(space='world')

        # Ensures that we aren't trying to calculate the first frame without another value to calculate
        if previous_value:
            # Finds the distance between the two foot positions of the current frame and previous frame
            current_distance = distance_between(current_value[0], current_value[1], current_value[2], previous_value[0], previous_value[1], previous_value[2])
            # Stores the distance difference
            all_values[x] = current_distance
        # Store the current value so we can compare it to the next
        previous_value = current_value

    # Add list to store all foot plant occurances
    foot_placement_dict[footstep_type] = []
    # Iterate through values starting at the first keyframe of the animation
    # Store any value lower than our "max distance threshold" that is less than the frame previous and after
    # (These values are the keyframes in which a foot is planted)
    for k,v in OrderedDict(all_values).iteritems():
        if k > 1 and k < len(all_values) - 1:
            if anim_type == 'walk':
                pm.currentTime(k)
                x_rotation = abs(foot.getRotation(space='world')[0])
                z_rotation = abs(foot.getRotation(space='world')[2])
                if v < all_values[k-1] and v < all_values[k+1] and v < threshold:
                    if x_rotation > 177 and x_rotation < 180 and z_rotation > 0 and z_rotation < 3:
                        foot_placement_dict[footstep_type].append(k)
                    elif x_rotation > 0 and x_rotation < 3 and z_rotation > 0 and z_rotation < 3:
                        foot_placement_dict[footstep_type].append(k)
```

All it took was a pretty small scrip.

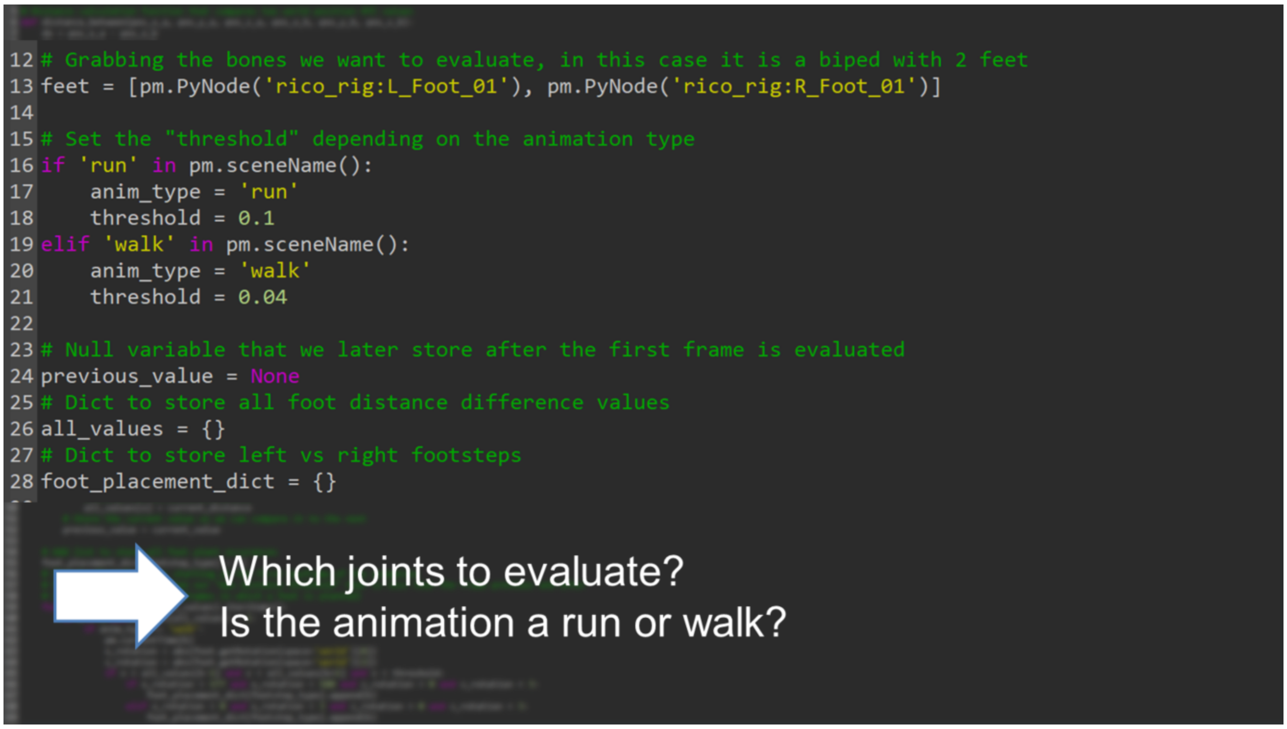
Don’t worry about trying to read this, I’m about to break this down so that you don’t have to try and read through this whole thing, especially if you aren’t a python coder.



```
# Distance calculation function that compares two world position XYZ values
def distance_between(pos_x_a, pos_y_a, pos_z_a, pos_x_b, pos_y_b, pos_z_b):
    dx = pos_x_a - pos_x_b
    dy = pos_y_a - pos_y_b
    dz = pos_z_a - pos_z_b
    return math.sqrt(dx*dx + dy*dy + dz*dz)
```

The first thing that we need to do is find the distance between the ankle joints within consecutive frames. For example, let’s say you are looking at a run cycle and the left leg of a character is swinging back down to hit the ground. On frame 7 you grab the position of the left ankle joint in world space, then you grab the world space position of the left ankle joint on frame 8, and you calculate the distance apart that the ankle joint has traveled from frame to frame.

Here is a simple function that can return the difference in distance of two points.

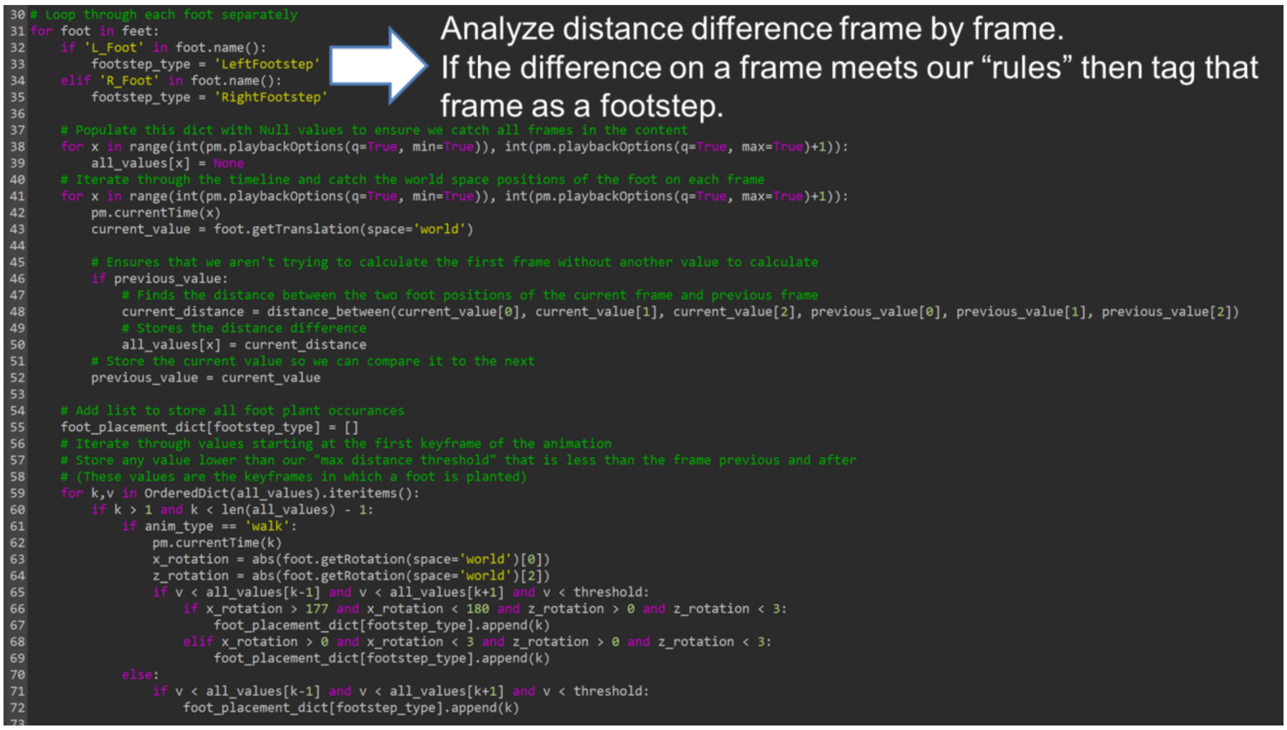


```
# Grabbing the bones we want to evaluate, in this case it is a biped with 2 feet
feet = [pm.PyNode('rico_rig:L_Foot_01'), pm.PyNode('rico_rig:R_Foot_01')]

# Set the "threshold" depending on the animation type
if 'run' in pm.sceneName():
    anim_type = 'run'
    threshold = 0.1
elif 'walk' in pm.sceneName():
    anim_type = 'walk'
    threshold = 0.04

# Null variable that we later store after the first frame is evaluated
previous_value = None
# Dict to store all foot distance difference values
all_values = {}
# Dict to store left vs right footsteps
foot_placement_dict = {}
```

Here we are just saying we want to evaluate the left and right foot joints and then depending on an animation being a walk or a run, there's a predetermined threshold that is slightly different based on the fact that the distance each foot moves depending on the gait of the character locomotion is pretty different. If you look at a run cycle, the ankle joint will have a further distance between frames than a walk cycle.

So what did this threshold do?

We know that when the foot is actually planted, it's position doesn't change a whole lot from frame to frame. This value simply says, do not bother even looking at anything greater than this. It was sort of a hacky but efficient way to filter out any obvious non-planted values, as sometimes those can otherwise mistakenly get tagged based on the character's leg movement when not planted.



Analyze distance difference frame by frame.
If the difference on a frame meets our “rules” then tag that frame as a footstep.

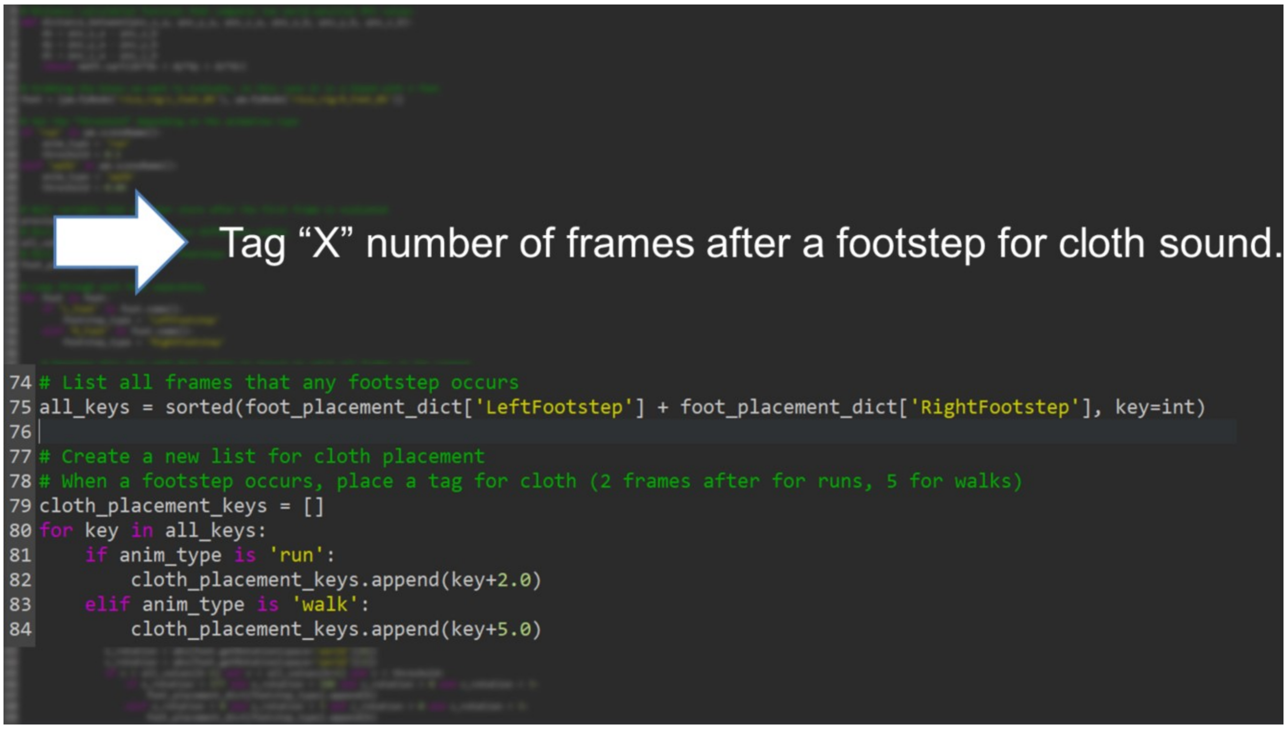
```python
# Loop through each foot separately
for foot in feet:
    if 'L_Foot' in foot.name():
        footstep_type = 'LeftFootstep'
    elif 'R_Foot' in foot.name():
        footstep_type = 'RightFootstep'

    # Populate this dict with Null values to ensure we catch all frames in the content
    for x in range(int(pm.playbackOptions(q=True, min=True)), int(pm.playbackOptions(q=True, max=True)+1)):
        all_values[x] = None
    # Iterate through the timeline and catch the world space positions of the foot on each frame
    for x in range(int(pm.playbackOptions(q=True, min=True)), int(pm.playbackOptions(q=True, max=True)+1)):
        pm.currentTime(x)
        current_value = foot.getTranslation(space='world')

        # Ensures that we aren't trying to calculate the first frame without another value to calculate
        if previous_value:
            # Finds the distance between the two foot positions of the current frame and previous frame
            current_distance = distance_between(current_value[0], current_value[1], current_value[2], previous_value[0], previous_value[1], previous_value[2])
            # Stores the distance difference
            all_values[x] = current_distance
        # Store the current value so we can compare it to the next
        previous_value = current_value

    # Add list to store all foot plant occurances
    foot_placement_dict[footstep_type] = []
    # Iterate through values starting at the first keyframe of the animation
    # Store any value lower than our "max distance threshold" that is less than the frame previous and after
    # (These values are the keyframes in which a foot is planted)
    for k,v in OrderedDict(all_values).iteritems():
        if k > 1 and k < len(all_values) - 1:
            if anim_type == 'walk':
                pm.currentTime(k)
                x_rotation = abs(foot.getRotation(space='world')[0])
                z_rotation = abs(foot.getRotation(space='world')[2])
                if v < all_values[k-1] and v < all_values[k+1] and v < threshold:
                    if x_rotation > 177 and x_rotation < 180 and z_rotation > 0 and z_rotation < 3:
                        foot_placement_dict[footstep_type].append(k)
                    elif x_rotation > 0 and x_rotation < 3 and z_rotation > 0 and z_rotation < 3:
                        foot_placement_dict[footstep_type].append(k)
            else:
                if v < all_values[k-1] and v < all_values[k+1] and v < threshold:
                    foot_placement_dict[footstep_type].append(k)
```

And here’s the bulk of the script that takes these factors into account.

We analyze twice here, one time for each foot. The analyzation goes through the timeline, finds the current value of the world space position of the foot on each frame, and compares that value with the world space position of the foot from the previous frame.

If a distance value is less than the frame before as well as less than the frame after we tag that frame as “the foot is planted.” By tagging the smallest value of movement in a particular range of values, we assume that the script has found a planted foot.

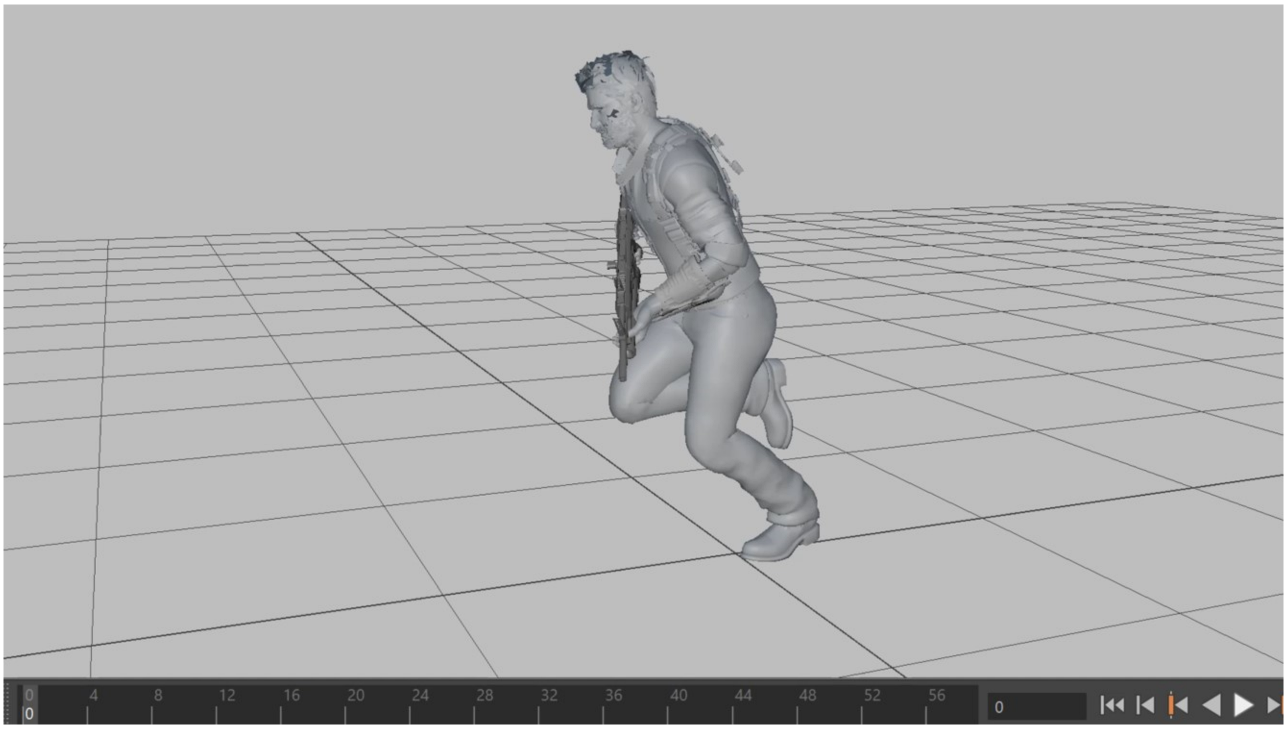
For walks, we also added a filter that made sure a foot wasn’t too drastically rotated up and down or twisted since the slower nature of walks meant that distances of the position of the feet weren’t as drastic frame to frame, and surveying both the pitch and roll of the foot gave us another good indication that a foot was planted as it ensured that the foot itself was flat on the ground.



```python
# List all frames that any footstep occurs
all_keys = sorted(foot_placement_dict['LeftFootstep'] + foot_placement_dict['RightFootstep'], key=int)

# Create a new list for cloth placement
# When a footstep occurs, place a tag for cloth (2 frames after for runs, 5 for walks)
cloth_placement_keys = []
for key in all_keys:
    if anim_type is 'run':
        cloth_placement_keys.append(key+2.0)
    elif anim_type is 'walk':
        cloth_placement_keys.append(key+5.0)
```

Another really nice thing we could do from here is take the footstep data, and depending on if an animation is a walk or a run, determine when you'd hear a cloth sound.

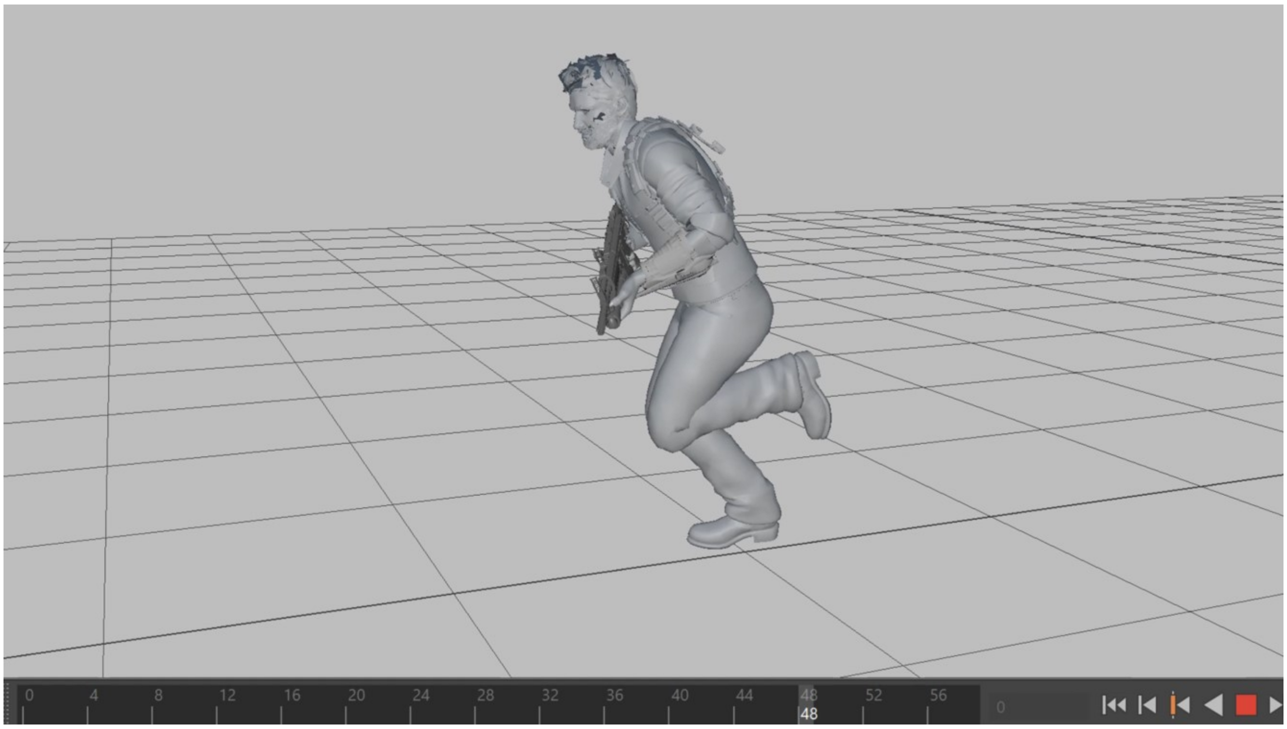
The character almost always hits the point where their legs cross each other after a step, at the same time on both runs or walks. For runs this was 2 frames after a step and for walks it was 5. Thanks to this consistent behavior we can automatically gather cloth sound triggers and not only are we gathering footstep data, but cloth sound data. This made the audio team extra happy as they now did not have to worry about manually tagging this in-game themselves…and they were also already benefitting from the automatically tagged footsteps, as was VFX too. So hey, we just made animators, the sound team, AND the VFX team very happy with all this automation.



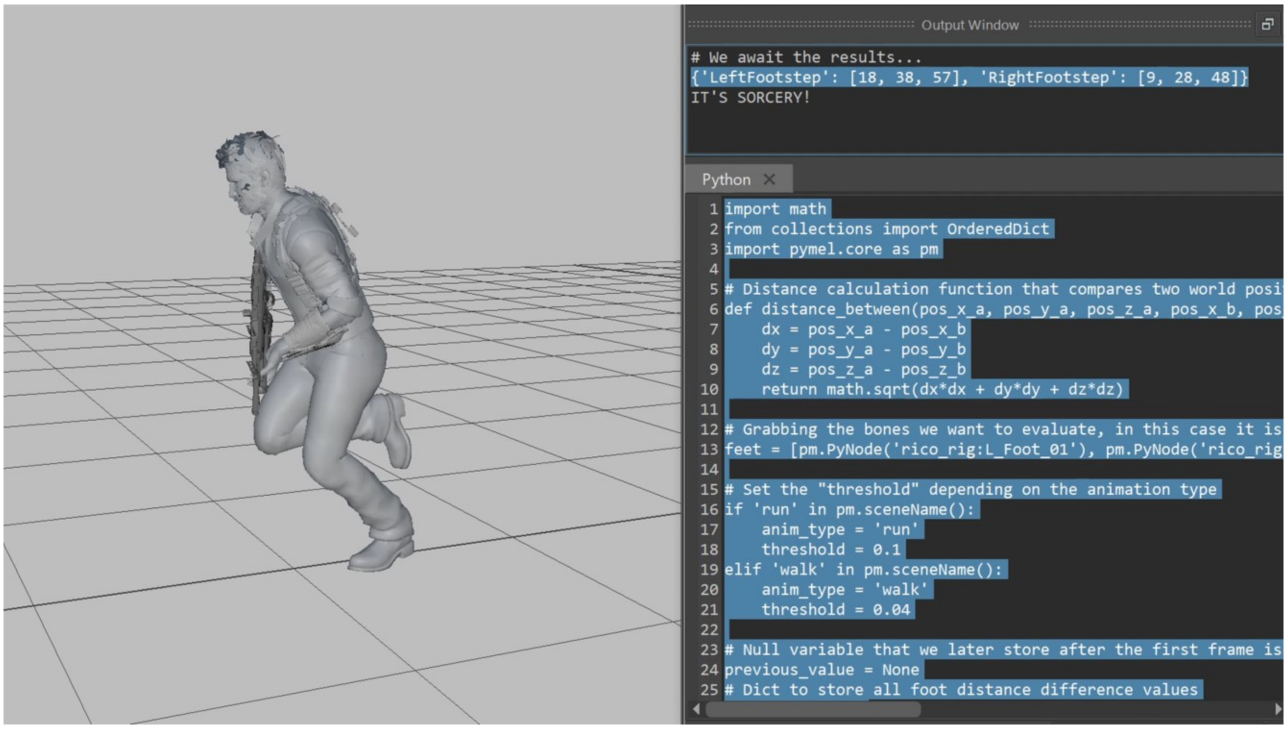
To the surprise of the animation, vfx, AND audio teams, and honestly myself…this method had a good success rate, I'd say 85-90% of footsteps and cloth were tagged correctly just by this. The rest was manually fixed, as sometimes extra footstep tags snuck their way in or a tag here or there was missed. However the amount of work to do that cleanup was so much less than what anyone had to do before when we were manually tagging footsteps.

So now it's time to see this in action.

We have ourselves a run animation.



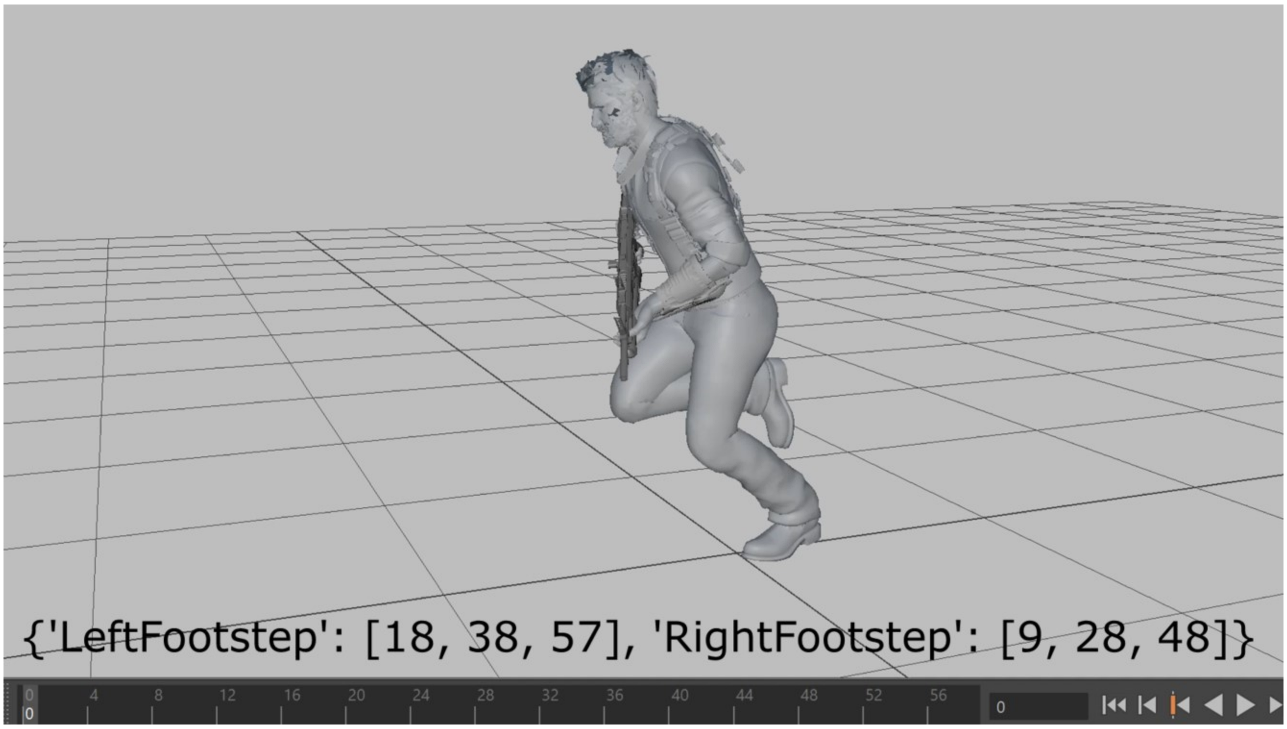
By quickly scrubbing we can see where the foot is planted.



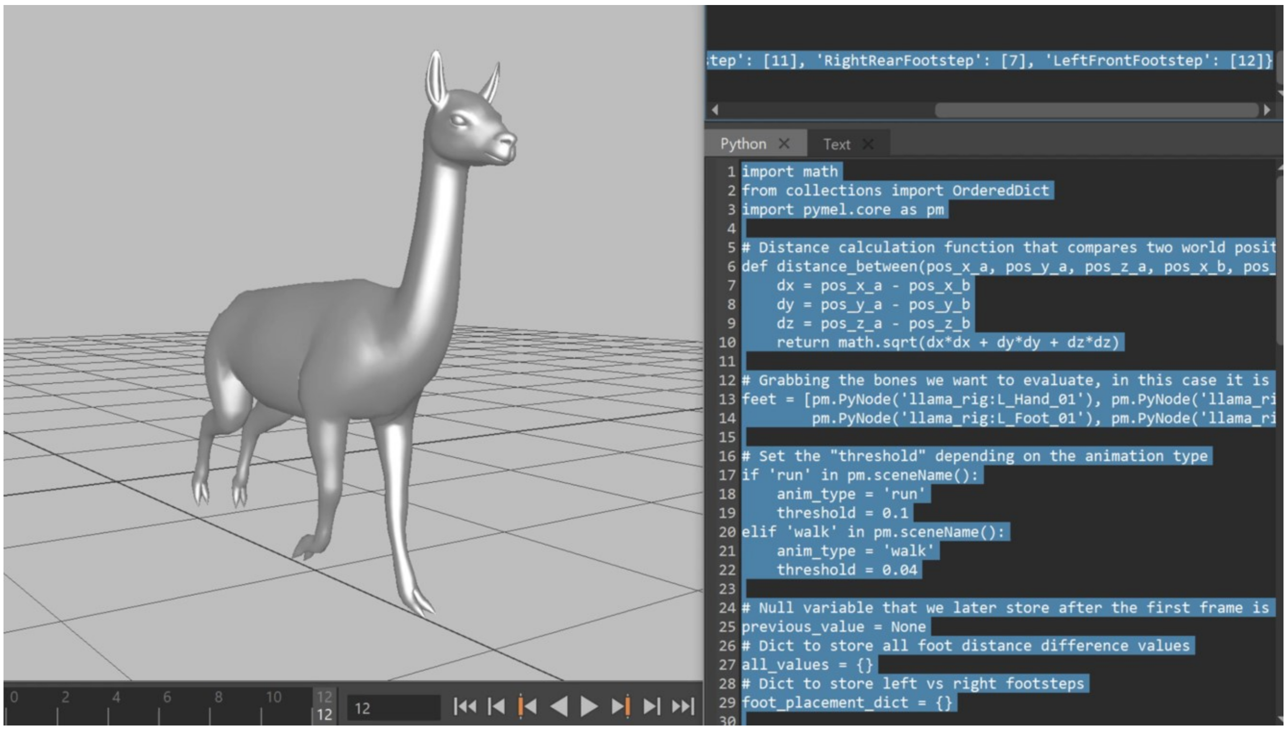
```
# We await the results...
{'LeftFootstep': [18, 38, 57], 'RightFootstep': [9, 28, 48]}
IT'S SORCERY!
```

```
import math
from collections import OrderedDict
import pymel.core as pm

# Distance calculation function that compares two world posi
def distance_between(pos_x_a, pos_y_a, pos_z_a, pos_x_b, pos
    dx = pos_x_a - pos_x_b
    dy = pos_y_a - pos_y_b
    dz = pos_z_a - pos_z_b
    return math.sqrt(dx*dx + dy*dy + dz*dz)

# Grabbing the bones we want to evaluate, in this case it is
feet = [pm.PyNode('rico_rig:L_Foot_01'), pm.PyNode('rico_rig

# Set the "threshold" depending on the animation type
if 'run' in pm.sceneName():
    anim_type = 'run'
    threshold = 0.1
elif 'walk' in pm.sceneName():
    anim_type = 'walk'
    threshold = 0.04

# Null variable that we later store after the first frame is
previous_value = None
# Dict to store all foot distance difference values
```

Now we run the script and it cycles through the animation as it evaluates each foot. When it is done evaluating, we get our auto-generated footstep tags. Let’s see how accurate this really is.



```
{'LeftFootstep': [18, 38, 57], 'RightFootstep': [9, 28, 48]}
```

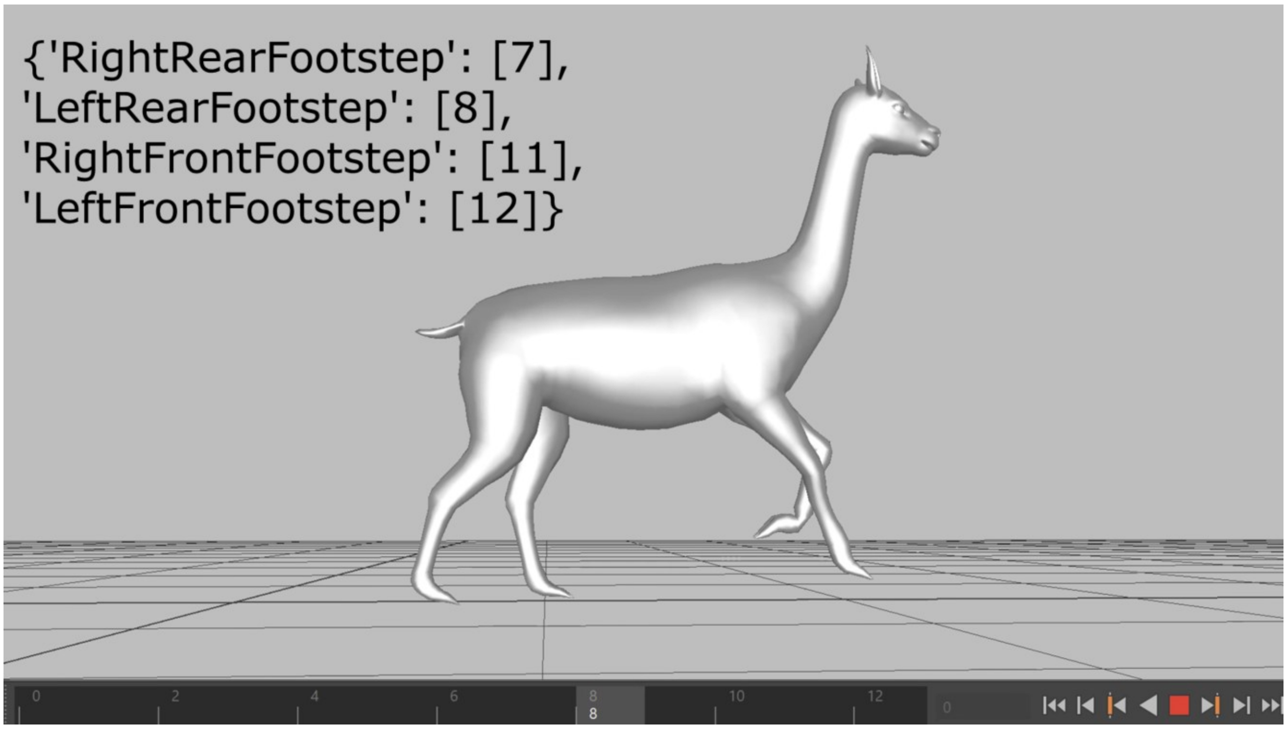


```
tep': [11], 'RightRearFootstep': [7], 'LeftFrontFootstep': [12]}
```

```python
import math
from collections import OrderedDict
import pymel.core as pm

# Distance calculation function that compares two world posit
def distance_between(pos_x_a, pos_y_a, pos_z_a, pos_x_b, pos_
    dx = pos_x_a - pos_x_b
    dy = pos_y_a - pos_y_b
    dz = pos_z_a - pos_z_b
    return math.sqrt(dx*dx + dy*dy + dz*dz)

# Grabbing the bones we want to evaluate, in this case it is
feet = [pm.PyNode('llama_rig:L_Hand_01'), pm.PyNode('llama_ri
        pm.PyNode('llama_rig:L_Foot_01'), pm.PyNode('llama_ri

# Set the "threshold" depending on the animation type
if 'run' in pm.sceneName():
    anim_type = 'run'
    threshold = 0.1
elif 'walk' in pm.sceneName():
    anim_type = 'walk'
    threshold = 0.04

# Null variable that we later store after the first frame is
previous_value = None
# Dict to store all foot distance difference values
all_values = {}
# Dict to store left vs right footsteps
foot_placement_dict = {}
```

And just for fun, let’s evaluate 4 feet and run this script on a llama the newest animal heroes in the Just Cause series.

Here’s our auto-generated data now…



```
{'RightRearFootstep': [7],
'LeftRearFootstep': [8],
'RightFrontFootstep': [11],
'LeftFrontFootstep': [12]}
```

And let’s check it out.

Oh daaaaang, that’s pretty good.

MOBU

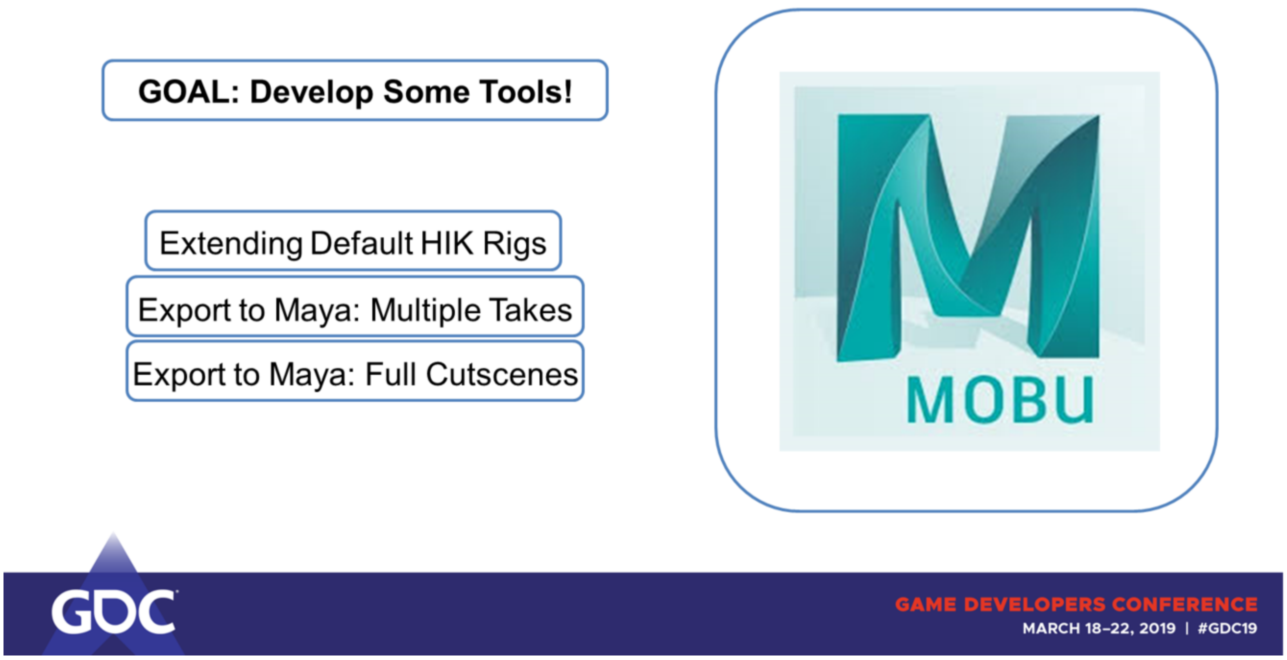
GDC

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

Moving over to MoBu now, here's what we had custom on JC3. Yep…nothing custom.

While we did have a way to send Maya content to MoBu and then from MoBu to Maya, that entire process on the MoBu side of things only involved making sure that you save your MoBu file with the take you want to transfer over to Maya saved as the active one.

The only MoBu specific tools we had were really just Maya tools that grabbed FBX files and transferred animation onto our Maya rigs, or would bake down and export animation content in Maya out to FBX.



GOAL: Develop Some Tools!

Extending Default HIK Rigs

Export to Maya: Multiple Takes

Export to Maya: Full Cutscenes

MOBU

GDC

GAME DEVELOPERS CONFERENCE

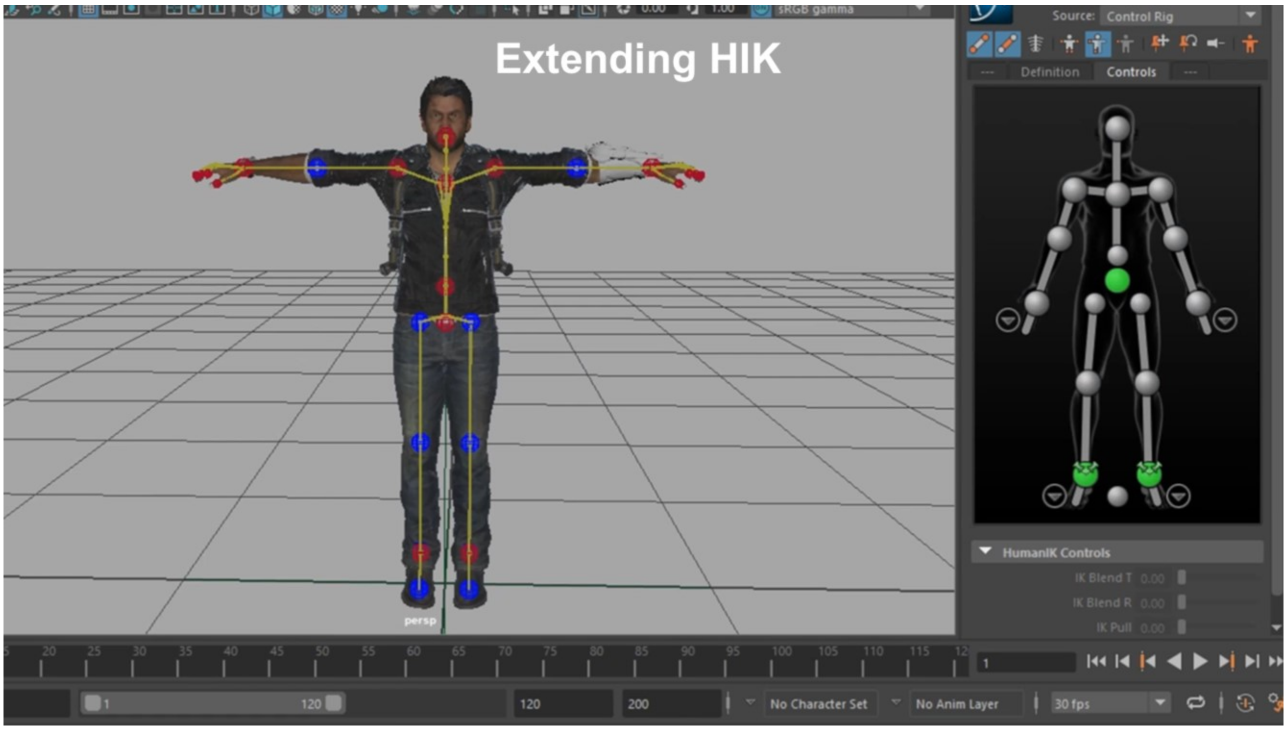
MARCH 18–22, 2019 | #GDC19

Our goal for MoBu was to actually make some tools! Now, we did already have some animation tools that my counterparts in Sweden have developed over the years, but there was nothing that existed for project and workflow specific needs for us on Just Cause.

So now that we were developing actual MoBu tools and allowing animators to use it more as a standalone tool, we wanted to extend the default HIK rigs to allow for more from within MoBu in terms of things such as retaining root motion data, weapon/attach data, and any other reference aside from the default HIK controls.

We also needed to drastically improve the process of sending content from MoBu to Maya.

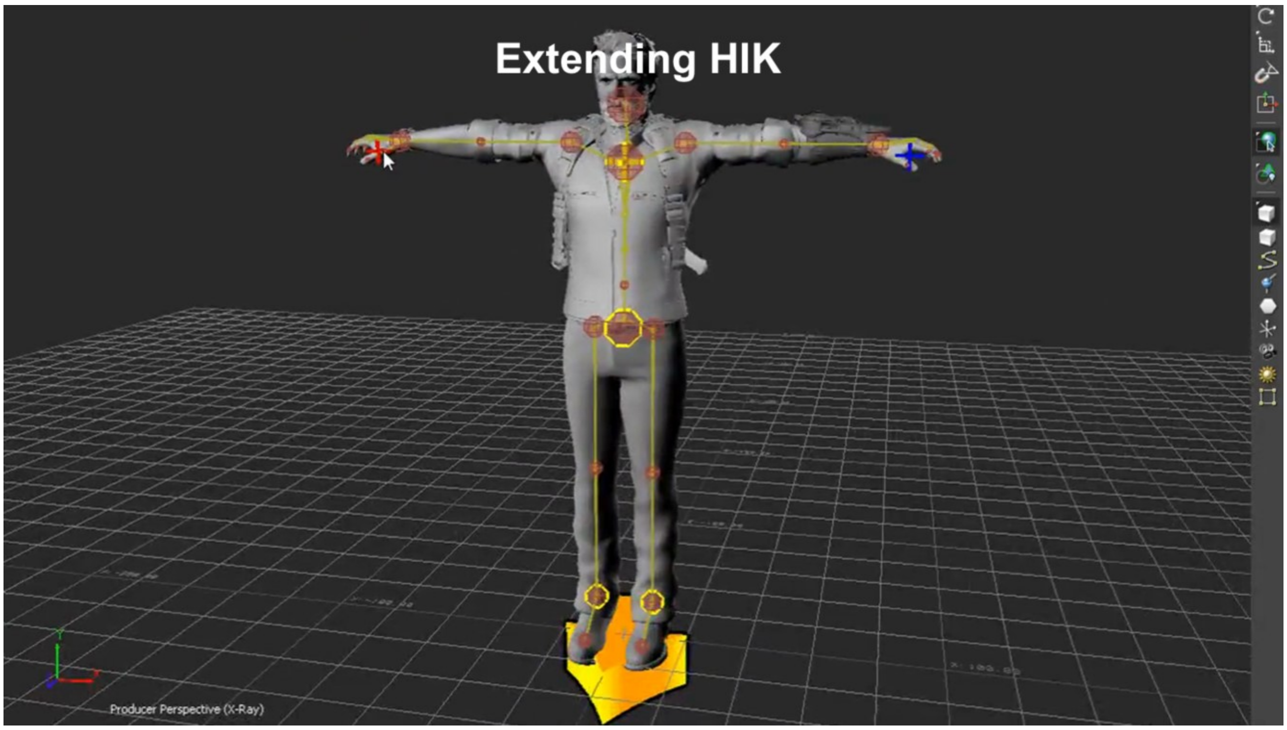
It became necessary to send multiple takes from within a MotionBuilder file over to Maya which handled animation content on a 1 animation per file structure. In addition to multiple takes we also had to send full cinematic sequences from Motion Builder to Maya, just as we had to do the same from Maya to MoBu. I'll detail this more later when I also talk about the cinematics.



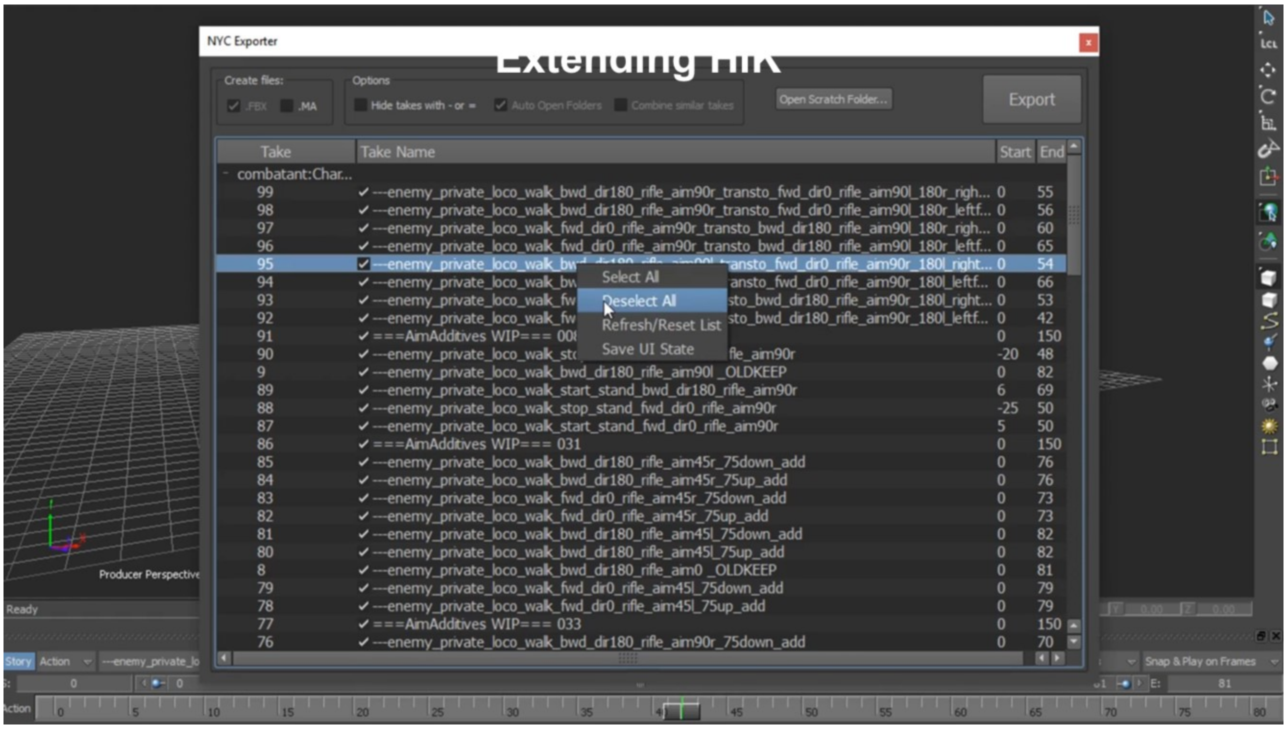
What we had on JC3 was the basic default HIK rig with no additional features. What we needed for JC4 was to add extensions to the HIK rigs so that we could pull that data, retain it, and modify if necessary.

We actually start off in Maya with the A-Pose character, t-pose it, and characterize it. Now this is our base to work on. From here we either build a Maya rig or an HIK rig from within Maya that we'll use in MotionBuilder.

The Maya rig gets saved off as an MA, and the HIK gets exported as an FBX file.



The FBX generated in Maya is then imported into motionbuilder and we run an auto-build of sorts that creates motionbuilder rig extensions to the root bone and some key reference joints that we use for character aiming and weapon or prop hand attachments. This process also adds a namespace and ensures that things that the Maya to MoBu conversion misses such as hand and foot contact values are properly set.

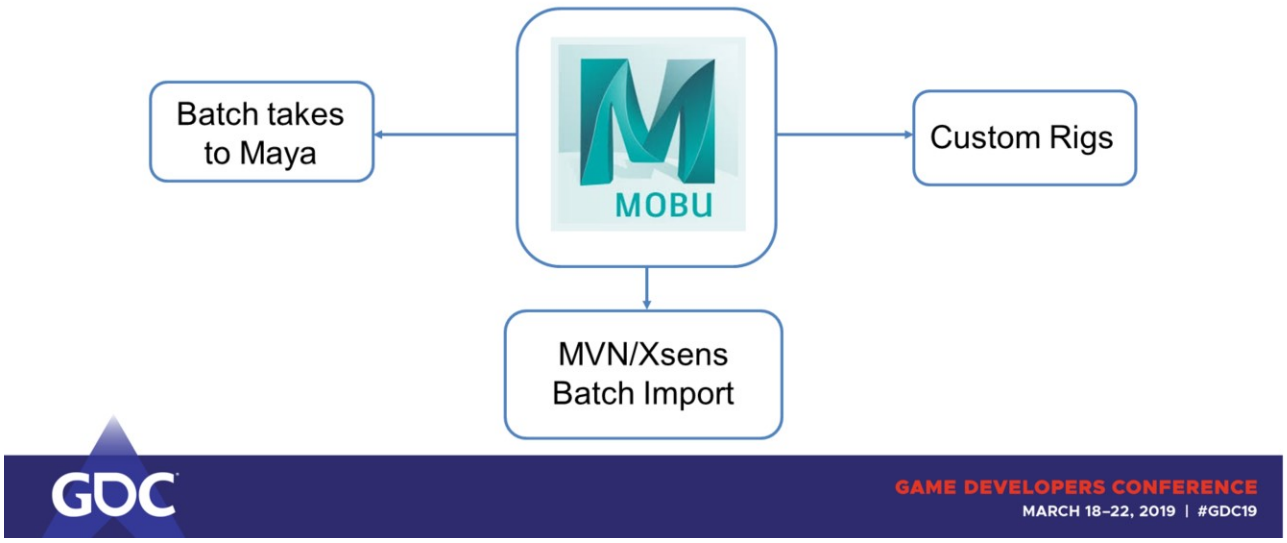


Create files: ✓ .FBX ☐ .MA

Options: ☐ Hide takes with - or = ✓ Auto Open Folders ☐ Combine similar takes

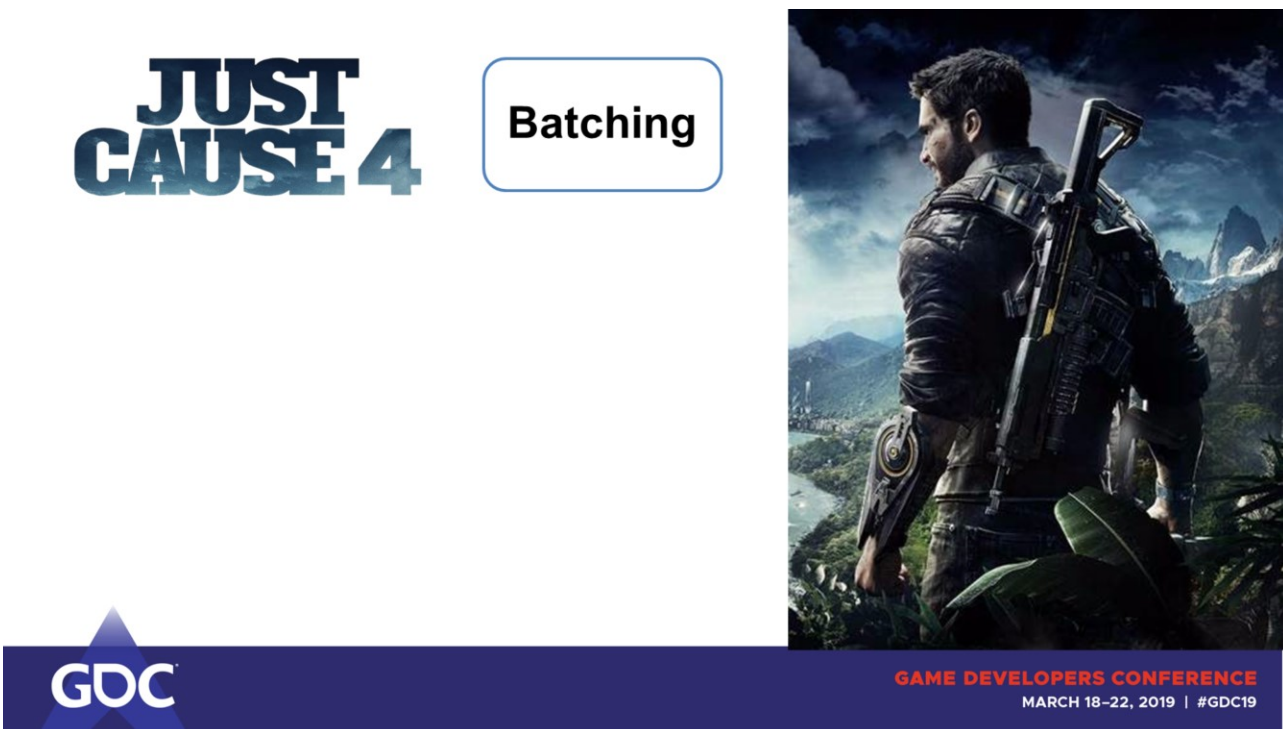
Take	Take Name	Start	End
combatant:Char...			
99	✓ ---enemy_private_loco_walk_bwd_dir180_rifle_aim90r_transto_fwd_dir0_rifle_aim90l_180r_righ...	0	55
98	✓ ---enemy_private_loco_walk_bwd_dir180_rifle_aim90r_transto_fwd_dir0_rifle_aim90l_180r_leftf...	0	56
97	✓ ---enemy_private_loco_walk_fwd_dir0_rifle_aim90r_transto_bwd_dir180_rifle_aim90l_180r_righ...	0	60
96	✓ ---enemy_private_loco_walk_fwd_dir0_rifle_aim90r_transto_bwd_dir180_rifle_aim90l_180r_leftf...	0	65
95	✓ ---enemy_private_loco_walk_bw [illegible] transto_fwd_dir0_rifle_aim90r_180l_right...	0	54
94	✓ ---enemy_private_loco_walk_bw [illegible] ansto_fwd_dir0_rifle_aim90r_180l_leftf...	0	66
93	✓ ---enemy_private_loco_walk_fw [illegible] sto_bwd_dir180_rifle_aim90r_180l_right...	0	53
92	✓ ---enemy_private_loco_walk_fw [illegible] sto_bwd_dir180_rifle_aim90r_180l_leftf...	0	42
91	✓ ===AimAdditives WIP=== 00 [illegible]	0	150
90	✓ ---enemy_private_loco_walk_st [illegible] fle_aim90r	-20	48
9	✓ ---enemy_private_loco_walk_bwd_dir180_rifle_aim90l _OLDKEEP	0	82
89	✓ ---enemy_private_loco_walk_start_stand_bwd_dir180_rifle_aim90r	6	69
88	✓ ---enemy_private_loco_walk_stop_stand_fwd_dir0_rifle_aim90r	-25	50
87	✓ ---enemy_private_loco_walk_start_stand_fwd_dir0_rifle_aim90r	5	50
86	✓ ===AimAdditives WIP=== 031	0	150
85	✓ ---enemy_private_loco_walk_bwd_dir180_rifle_aim45r_75down_add	0	76
84	✓ ---enemy_private_loco_walk_bwd_dir180_rifle_aim45r_75up_add	0	76
83	✓ ---enemy_private_loco_walk_fwd_dir0_rifle_aim45r_75down_add	0	73
82	✓ ---enemy_private_loco_walk_fwd_dir0_rifle_aim45r_75up_add	0	73
81	✓ ---enemy_private_loco_walk_bwd_dir180_rifle_aim45l_75down_add	0	82
80	✓ ---enemy_private_loco_walk_bwd_dir180_rifle_aim45l_75up_add	0	82
8	✓ ---enemy_private_loco_walk_bwd_dir180_rifle_aim0 _OLDKEEP	0	81
79	✓ ---enemy_private_loco_walk_fwd_dir0_rifle_aim45l_75down_add	0	79
78	✓ ---enemy_private_loco_walk_fwd_dir0_rifle_aim45l_75up_add	0	79
77	✓ ===AimAdditives WIP=== 033	0	150
76	✓ ---enemy_private_loco_walk_bwd_dir180_rifle_aim90r_75down_add	0	70

Now that the team has a custom MoBu rig that they're animating with, they need to be able to push that content over to Maya. On JC3 animators would have to save their MoBu file with the take they wanted to transfer to Maya as the active take. Since animators were utilizing MoBu much more this time around, and especially for the animator on our team that ONLY used MoBu, this wasn't going to cut it. Some files could contain many takes such as this one, so we needed a way to quickly get any and all content out of MoBu, so we developed this UI that allows you to pick and choose whatever animations you want…which then get plotted down and completely stripped down so that the result is a simple skeleton that is able to be very quickly brought into Maya for transfer to the Maya rigs.



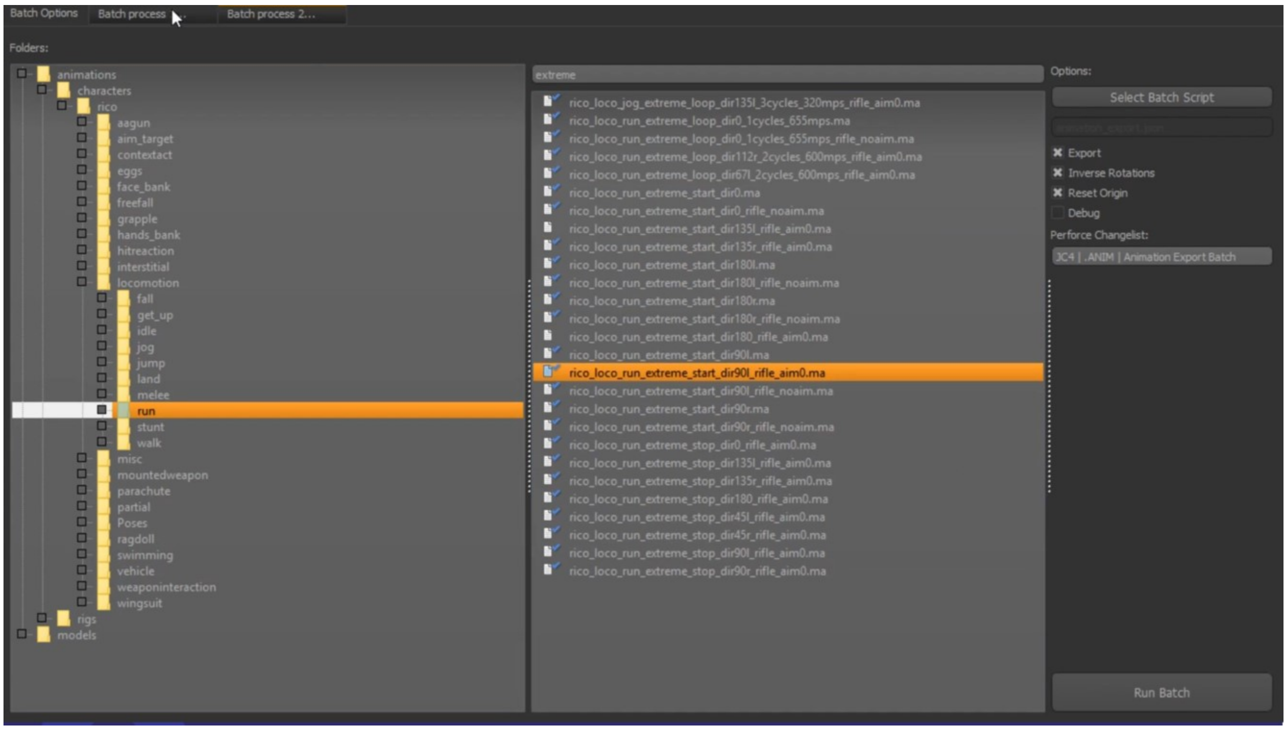
It may not be a ton, but these changes did make MoBu much more accessible as more of a standalone DCC for the team.

Also, I do have it listed here because it's worth mentioning. There is a batch import tool for Xsens content into MoBu, but it's literally just a button that asks what directory you want to import from so for sake of that visually not being too interesting, I'm just listing it and telling you now…we have this.


JUST CAUSE 4
GDC
GAME DEVELOPERS CONFERENCE
MARCH 18-22, 2019 | #GDC19

Batching

This section isn't quite as long as the others but it was really at the core of making everything quicker and more automated for both the tech animators and animators.



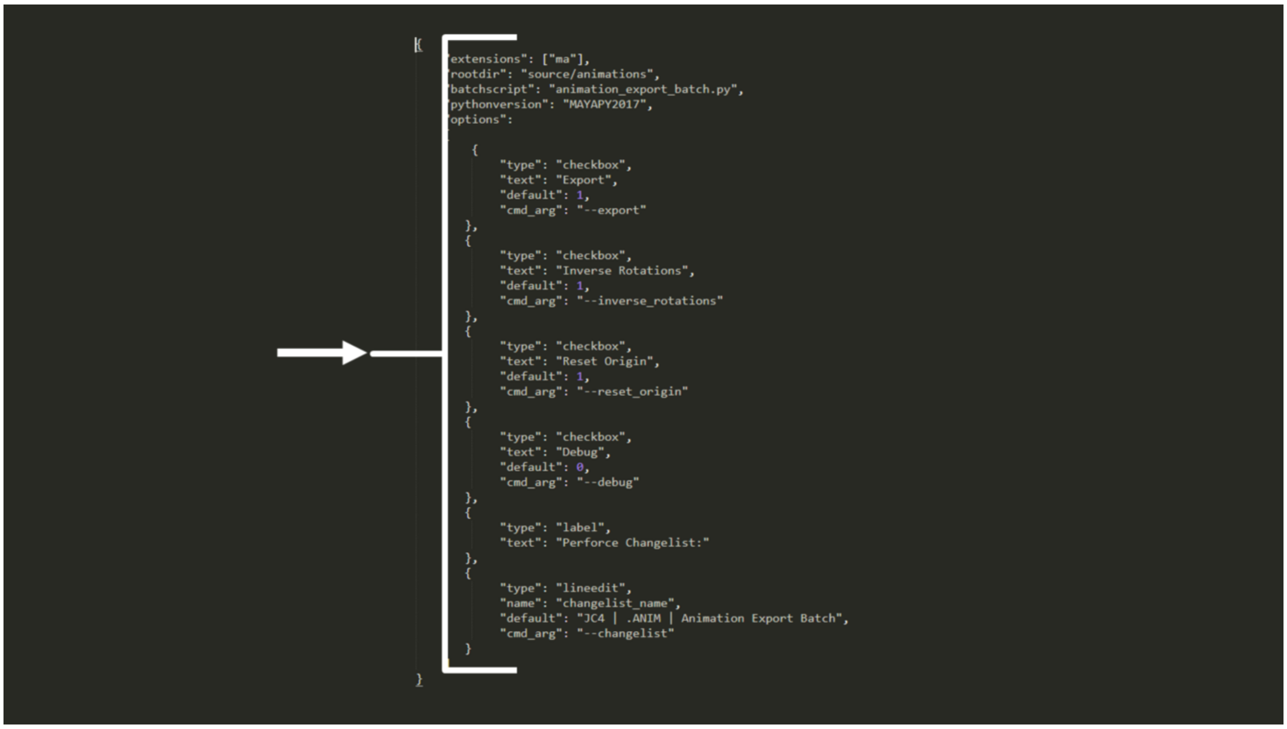
```
animations
  characters
    rico
      aagun
      aim_target
      contextact
      eggs
      face_bank
      freefall
      grapple
      hands_bank
      hitreaction
      interstitial
      locomotion
        fall
        get_up
        idle
        jog
        jump
        land
        melee
        run
        stunt
        walk
      misc
      mountedweapon
      parachute
      partial
      Poses
      ragdoll
      swimming
      vehicle
      weaponinteraction
      wingsuit
  rigs
models
```

```
extreme
rico_loco_jog_extreme_loop_dir135l_3cycles_320mps_rifle_aim0.ma
rico_loco_run_extreme_loop_dir0_1cycles_655mps.ma
rico_loco_run_extreme_loop_dir0_1cycles_655mps_rifle_noaim.ma
rico_loco_run_extreme_loop_dir112r_2cycles_600mps_rifle_aim0.ma
rico_loco_run_extreme_loop_dir67l_2cycles_600mps_rifle_aim0.ma
rico_loco_run_extreme_start_dir0.ma
rico_loco_run_extreme_start_dir0_rifle_noaim.ma
rico_loco_run_extreme_start_dir135l_rifle_aim0.ma
rico_loco_run_extreme_start_dir135r_rifle_aim0.ma
rico_loco_run_extreme_start_dir180l.ma
rico_loco_run_extreme_start_dir180l_rifle_noaim.ma
rico_loco_run_extreme_start_dir180r.ma
rico_loco_run_extreme_start_dir180r_rifle_noaim.ma
rico_loco_run_extreme_start_dir180_rifle_aim0.ma
rico_loco_run_extreme_start_dir90l.ma
rico_loco_run_extreme_start_dir90l_rifle_aim0.ma
rico_loco_run_extreme_start_dir90l_rifle_noaim.ma
rico_loco_run_extreme_start_dir90r.ma
rico_loco_run_extreme_start_dir90r_rifle_noaim.ma
rico_loco_run_extreme_stop_dir0_rifle_aim0.ma
rico_loco_run_extreme_stop_dir135l_rifle_aim0.ma
rico_loco_run_extreme_stop_dir135r_rifle_aim0.ma
rico_loco_run_extreme_stop_dir180_rifle_aim0.ma
rico_loco_run_extreme_stop_dir45l_rifle_aim0.ma
rico_loco_run_extreme_stop_dir45r_rifle_aim0.ma
rico_loco_run_extreme_stop_dir90l_rifle_aim0.ma
rico_loco_run_extreme_stop_dir90r_rifle_aim0.ma
```

✓ Export
✓ Inverse Rotations
✓ Reset Origin
☐ Debug

Perforce Changelist: JC4 | .ANIM | Animation Export Batch

This batch processing tool became our central core to running heavy processes into and out of Maya. EVERYTHING was streamed through here. The way this works is that the user selects a batch process from a list of json files that we've made for them. This could be an animation export, or a mobu to maya transfer. When they select the file, any options pertaining to that file show up on the right. The user can tweak those settings as necessary. When you load a script, any directories with files that pertain to the script also pop up on the left. When you select a folder, the files within that directory show up for you to multi-select and run the batch on. When you run a batch, a tab pops up within the UI that runs a Maya standalone thread and batches your content headless. You are able to run multiple different processes at once this way without ever opening up Maya directly. Optimal for computer performance and speed of data processing. And very optimal for being 100% animator friendly as they do not have to worry about running anything from cmd line, and could potentially customize these batch options themselves.



```
{
    "extensions": ["ma"],
    "rootdir": "source/animations",
    "batchscript": "animation_export_batch.py",
    "pythonversion": "MAYAPY2017",
    "options":

        {
            "type": "checkbox",
            "text": "Export",
            "default": 1,
            "cmd_arg": "--export"
        },
        {
            "type": "checkbox",
            "text": "Inverse Rotations",
            "default": 1,
            "cmd_arg": "--inverse_rotations"
        },
        {
            "type": "checkbox",
            "text": "Reset Origin",
            "default": 1,
            "cmd_arg": "--reset_origin"
        },
        {
            "type": "checkbox",
            "text": "Debug",
            "default": 0,
            "cmd_arg": "--debug"
        },
        {
            "type": "label",
            "text": "Perforce Changelist:"
        },
        {
            "type": "lineedit",
            "name": "changelist_name",
            "default": "JC4 | .ANIM | Animation Export Batch",
            "cmd_arg": "--changelist"
        }
}
```

A quick look under the hood. We have a very simple json dict where we state some rules and list out all of our options for the user. In this case we're looking at our Maya animation exporter. We tell this file..

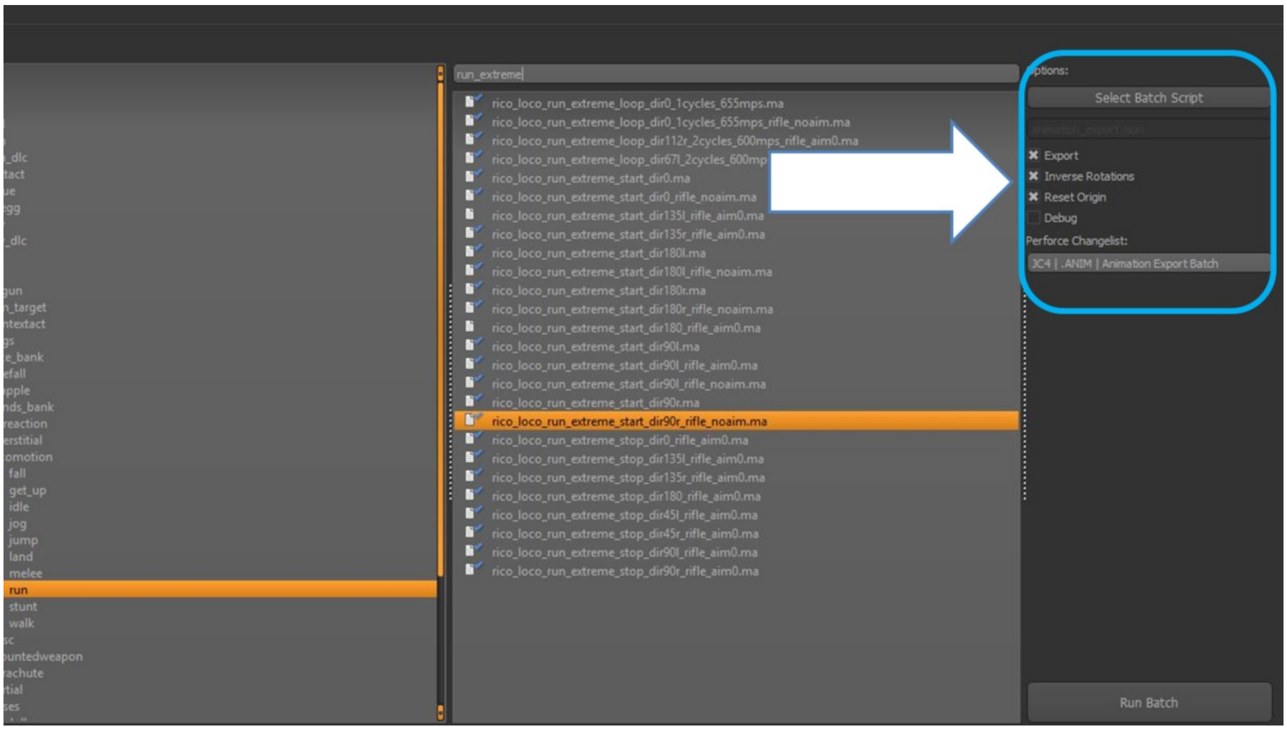
What filetypes to look for.

Which directory to look for them in.

Which batch process to actually run.

The version of Maya standalone to use.

And all of the options we want to list…

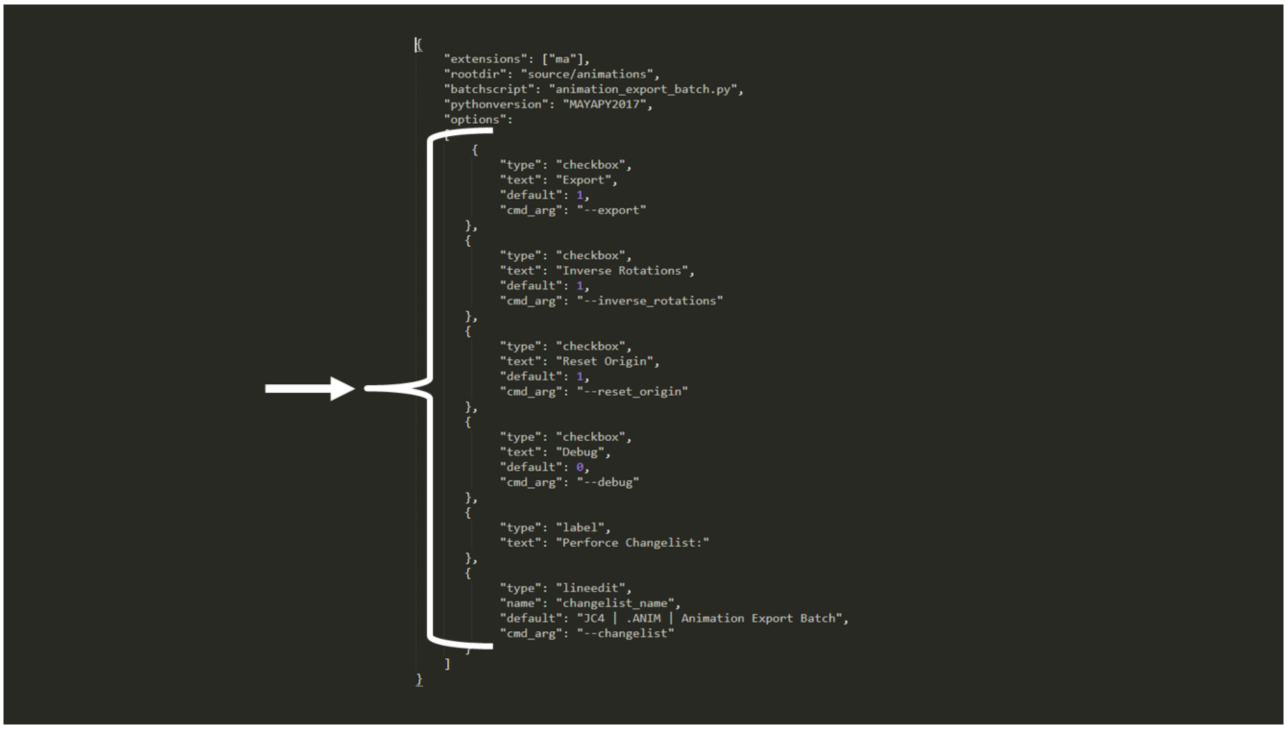


```
run_extreme
rico_loco_run_extreme_loop_dir0_1cycles_655mps.ma
rico_loco_run_extreme_loop_dir0_1cycles_655mps_rifle_noaim.ma
rico_loco_run_extreme_loop_dir112r_2cycles_600mps_rifle_aim0.ma
rico_loco_run_extreme_loop_dir67l_2cycles_600mp
rico_loco_run_extreme_start_dir0.ma
rico_loco_run_extreme_start_dir0_rifle_noaim.ma
rico_loco_run_extreme_start_dir135l_rifle_aim0.ma
rico_loco_run_extreme_start_dir135r_rifle_aim0.ma
rico_loco_run_extreme_start_dir180l.ma
rico_loco_run_extreme_start_dir180l_rifle_noaim.ma
rico_loco_run_extreme_start_dir180r.ma
rico_loco_run_extreme_start_dir180r_rifle_noaim.ma
rico_loco_run_extreme_start_dir180_rifle_aim0.ma
rico_loco_run_extreme_start_dir90l.ma
rico_loco_run_extreme_start_dir90l_rifle_aim0.ma
rico_loco_run_extreme_start_dir90l_rifle_noaim.ma
rico_loco_run_extreme_start_dir90r.ma
rico_loco_run_extreme_start_dir90r_rifle_noaim.ma
rico_loco_run_extreme_stop_dir0_rifle_aim0.ma
rico_loco_run_extreme_stop_dir135l_rifle_aim0.ma
rico_loco_run_extreme_stop_dir135r_rifle_aim0.ma
rico_loco_run_extreme_stop_dir180_rifle_aim0.ma
rico_loco_run_extreme_stop_dir45l_rifle_aim0.ma
rico_loco_run_extreme_stop_dir45r_rifle_aim0.ma
rico_loco_run_extreme_stop_dir90l_rifle_aim0.ma
rico_loco_run_extreme_stop_dir90r_rifle_aim0.ma
```

✓ Export
✓ Inverse Rotations
✓ Reset Origin
☐ Debug

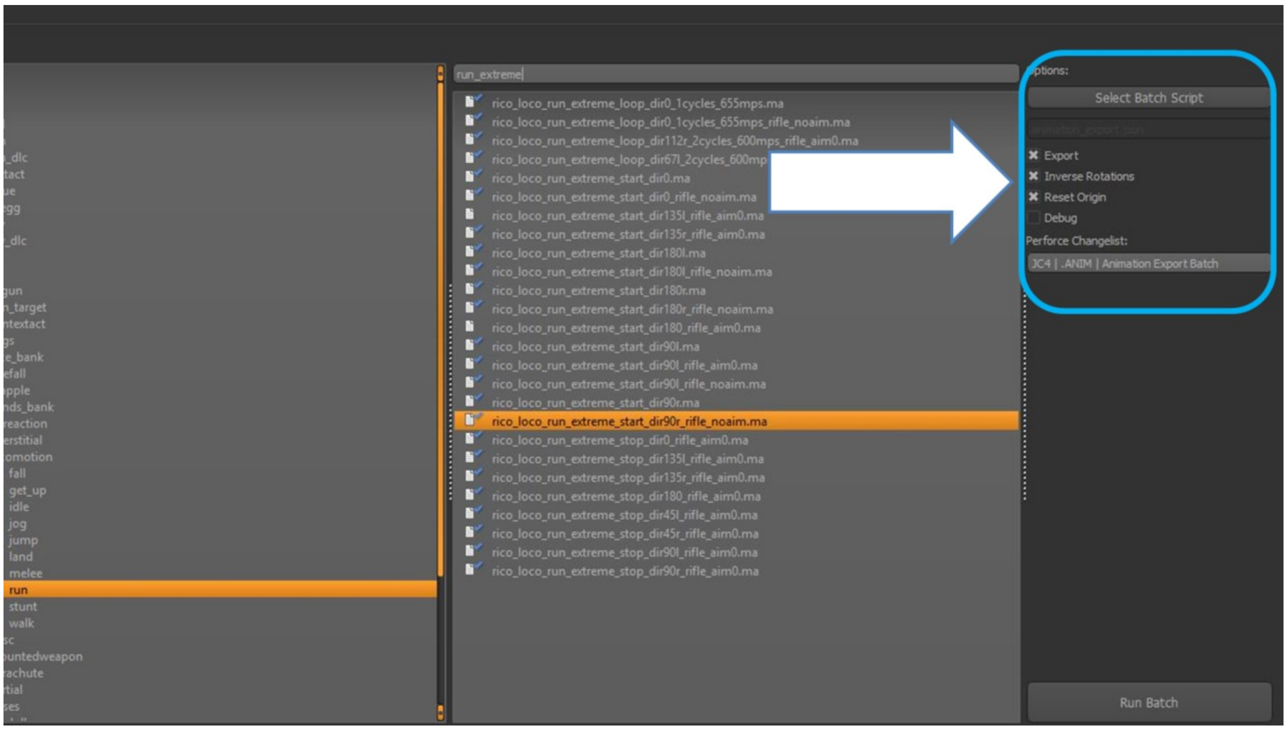
Perforce Changelist: JC4 | .ANIM | Animation Export Batch

Here.



```
{
    "extensions": ["ma"],
    "rootdir": "source/animations",
    "batchscript": "animation_export_batch.py",
    "pythonversion": "MAYAPY2017",
    "options":
    [
        {
            "type": "checkbox",
            "text": "Export",
            "default": 1,
            "cmd_arg": "--export"
        },
        {
            "type": "checkbox",
            "text": "Inverse Rotations",
            "default": 1,
            "cmd_arg": "--inverse_rotations"
        },
        {
            "type": "checkbox",
            "text": "Reset Origin",
            "default": 1,
            "cmd_arg": "--reset_origin"
        },
        {
            "type": "checkbox",
            "text": "Debug",
            "default": 0,
            "cmd_arg": "--debug"
        },
        {
            "type": "label",
            "text": "Perforce Changelist:"
        },
        {
            "type": "lineedit",
            "name": "changelist_name",
            "default": "JC4 | .ANIM | Animation Export Batch",
            "cmd_arg": "--changelist"
        }
    ]
}
```

Using as an example… animation exporting, these options let you create things such as toggles, for maybe turning on or off the generation of a debug animation file, or a line edit for customizing what you want to call the perforce changelist that all your animations get added to. We set the type of UI widget, what to label it, the default behavior, and the command that we will use to pass the state of each option through to the python batch file when running the process….



```
_dlc
tact
ue
egg
_dlc
gun
n_target
ntextact
gs
e_bank
efall
pple
nds_bank
reaction
erstitial
omotion
 fall
 get_up
 idle
 jog
 jump
 land
 melee
 run
 stunt
 walk
sc
ountedweapon
achute
tial
ses
```

```
run_extreme
rico_loco_run_extreme_loop_dir0_1cycles_655mps.ma
rico_loco_run_extreme_loop_dir0_1cycles_655mps_rifle_noaim.ma
rico_loco_run_extreme_loop_dir112r_2cycles_600mps_rifle_aim0.ma
rico_loco_run_extreme_loop_dir67l_2cycles_600mp
rico_loco_run_extreme_start_dir0.ma
rico_loco_run_extreme_start_dir0_rifle_noaim.ma
rico_loco_run_extreme_start_dir135l_rifle_aim0.ma
rico_loco_run_extreme_start_dir135r_rifle_aim0.ma
rico_loco_run_extreme_start_dir180l.ma
rico_loco_run_extreme_start_dir180l_rifle_noaim.ma
rico_loco_run_extreme_start_dir180r.ma
rico_loco_run_extreme_start_dir180r_rifle_noaim.ma
rico_loco_run_extreme_start_dir180_rifle_aim0.ma
rico_loco_run_extreme_start_dir90l.ma
rico_loco_run_extreme_start_dir90l_rifle_aim0.ma
rico_loco_run_extreme_start_dir90l_rifle_noaim.ma
rico_loco_run_extreme_start_dir90r.ma
rico_loco_run_extreme_start_dir90r_rifle_noaim.ma
rico_loco_run_extreme_stop_dir0_rifle_aim0.ma
rico_loco_run_extreme_stop_dir135l_rifle_aim0.ma
rico_loco_run_extreme_stop_dir135r_rifle_aim0.ma
rico_loco_run_extreme_stop_dir180_rifle_aim0.ma
rico_loco_run_extreme_stop_dir45l_rifle_aim0.ma
rico_loco_run_extreme_stop_dir45r_rifle_aim0.ma
rico_loco_run_extreme_stop_dir90l_rifle_aim0.ma
rico_loco_run_extreme_stop_dir90r_rifle_aim0.ma
```

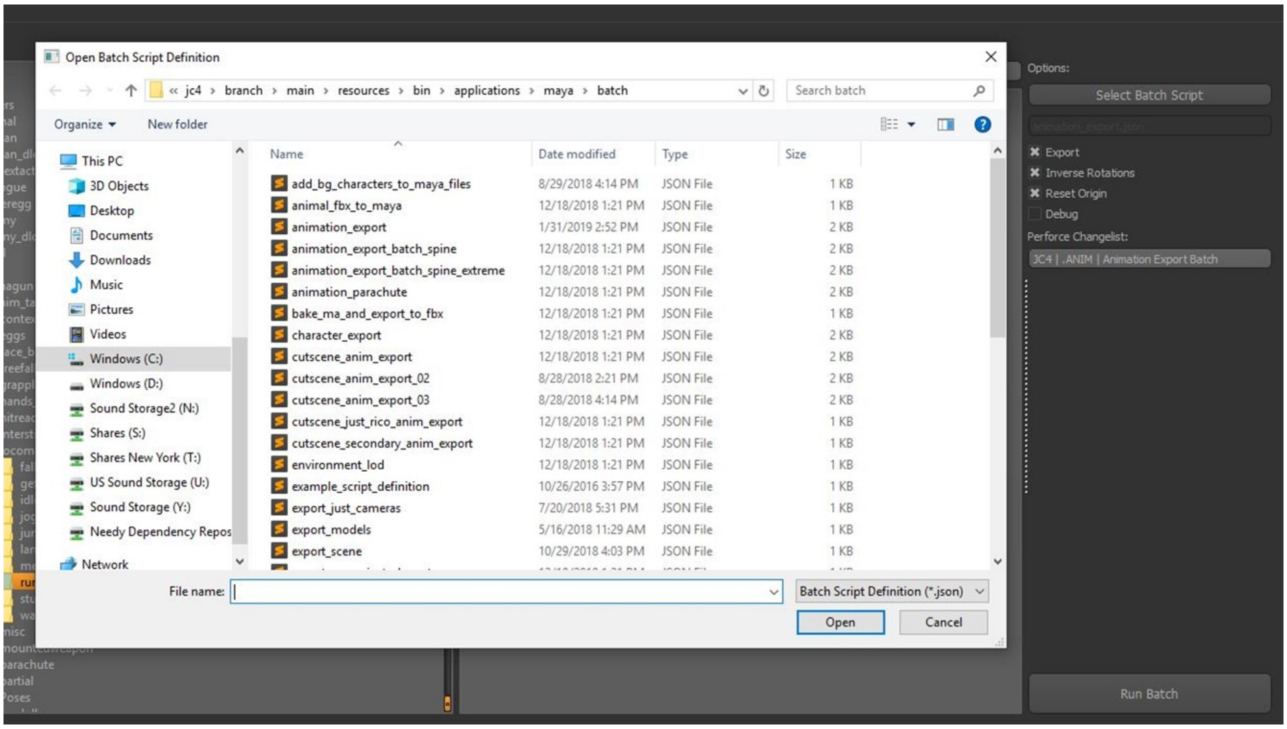
Options:

animation_export.json

- [✓] Export
- [✓] Inverse Rotations
- [✓] Reset Origin
- [ ] Debug

Perforce Changelist: JC4 | .ANIM | Animation Export Batch

Which magically populates your options here as seen a minute age, whenever you hit the select batch script button…



Name	Date modified	Type	Size
add_bg_characters_to_maya_files	8/29/2018 4:14 PM	JSON File	1 KB
animal_fbx_to_maya	12/18/2018 1:21 PM	JSON File	1 KB
animation_export	1/31/2019 2:52 PM	JSON File	2 KB
animation_export_batch_spine	12/18/2018 1:21 PM	JSON File	2 KB
animation_export_batch_spine_extreme	12/18/2018 1:21 PM	JSON File	2 KB
animation_parachute	12/18/2018 1:21 PM	JSON File	2 KB
bake_ma_and_export_to_fbx	12/18/2018 1:21 PM	JSON File	1 KB
character_export	12/18/2018 1:21 PM	JSON File	2 KB
cutscene_anim_export	12/18/2018 1:21 PM	JSON File	2 KB
cutscene_anim_export_02	8/28/2018 2:21 PM	JSON File	2 KB
cutscene_anim_export_03	8/28/2018 4:14 PM	JSON File	2 KB
cutscene_just_rico_anim_export	12/18/2018 1:21 PM	JSON File	1 KB
cutscene_secondary_anim_export	12/18/2018 1:21 PM	JSON File	1 KB
environment_lod	12/18/2018 1:21 PM	JSON File	1 KB
example_script_definition	10/26/2016 3:57 PM	JSON File	1 KB
export_just_cameras	7/20/2018 5:31 PM	JSON File	1 KB
export_models	5/16/2018 11:29 AM	JSON File	1 KB
export_scene	10/29/2018 4:03 PM	JSON File	1 KB

File name:

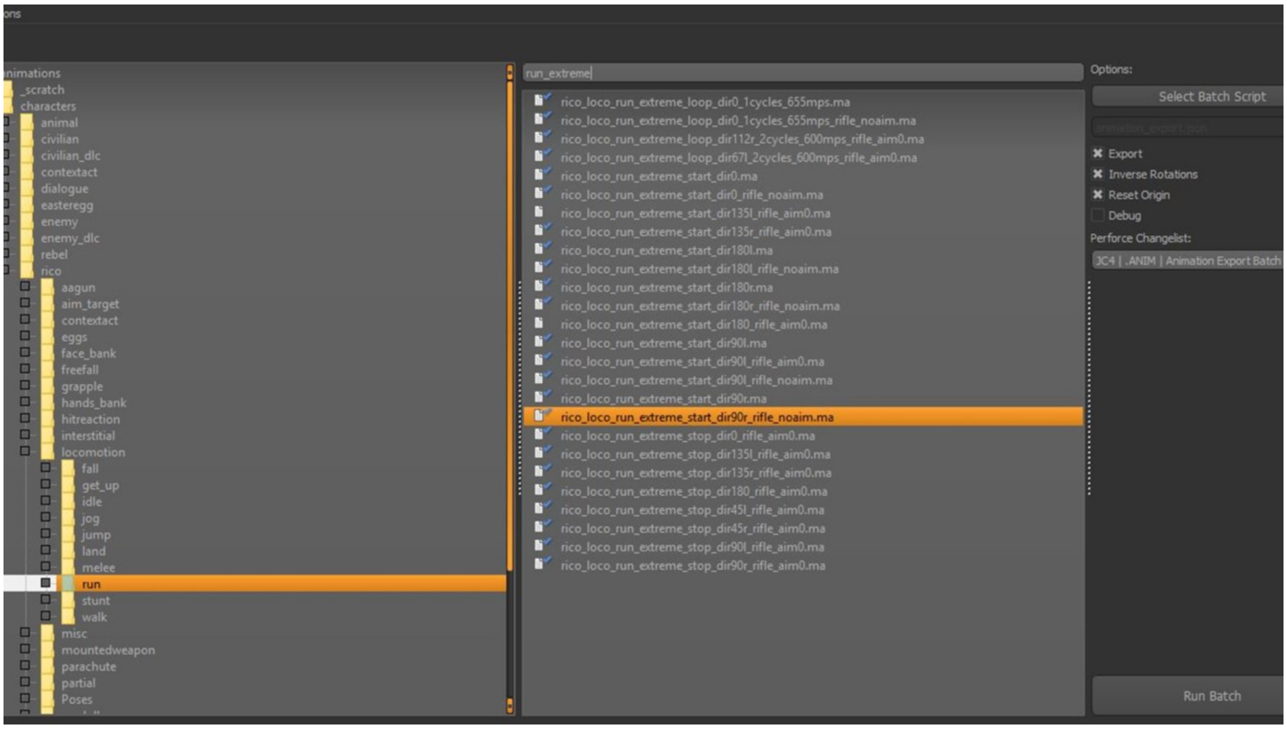
Batch Script Definition (*.json)

Options:

✖ Export
✖ Inverse Rotations
✖ Reset Origin
Debug

Perforce Changelist: JC4 | .ANIM | Animation Export Batch

And choose your specific batch. You can actually see here that I have quite a few batch scripts. Some of these are officially submitted into perforce for "official" use and others …like "cutscene_anim_export_02" and "03" here in the middle are just local processes I set up to test something out and maybe run a specific process that I only to do once or twice.



run_extreme

rico_loco_run_extreme_loop_dir0_1cycles_655mps.ma
rico_loco_run_extreme_loop_dir0_1cycles_655mps_rifle_noaim.ma
rico_loco_run_extreme_loop_dir112r_2cycles_600mps_rifle_aim0.ma
rico_loco_run_extreme_loop_dir67l_2cycles_600mps_rifle_aim0.ma
rico_loco_run_extreme_start_dir0.ma
rico_loco_run_extreme_start_dir0_rifle_noaim.ma
rico_loco_run_extreme_start_dir135l_rifle_aim0.ma
rico_loco_run_extreme_start_dir135r_rifle_aim0.ma
rico_loco_run_extreme_start_dir180l.ma
rico_loco_run_extreme_start_dir180l_rifle_noaim.ma
rico_loco_run_extreme_start_dir180r.ma
rico_loco_run_extreme_start_dir180r_rifle_noaim.ma
rico_loco_run_extreme_start_dir180_rifle_aim0.ma
rico_loco_run_extreme_start_dir90l.ma
rico_loco_run_extreme_start_dir90l_rifle_aim0.ma
rico_loco_run_extreme_start_dir90l_rifle_noaim.ma
rico_loco_run_extreme_start_dir90r.ma
rico_loco_run_extreme_start_dir90r_rifle_noaim.ma
rico_loco_run_extreme_stop_dir0_rifle_aim0.ma
rico_loco_run_extreme_stop_dir135l_rifle_aim0.ma
rico_loco_run_extreme_stop_dir135r_rifle_aim0.ma
rico_loco_run_extreme_stop_dir180_rifle_aim0.ma
rico_loco_run_extreme_stop_dir45l_rifle_aim0.ma
rico_loco_run_extreme_stop_dir45r_rifle_aim0.ma
rico_loco_run_extreme_stop_dir90l_rifle_aim0.ma
rico_loco_run_extreme_stop_dir90r_rifle_aim0.ma

Options:

✓ Export
✓ Inverse Rotations
✓ Reset Origin
☐ Debug

Perforce Changelist:
JC4 | .ANIM | Animation Export Batch

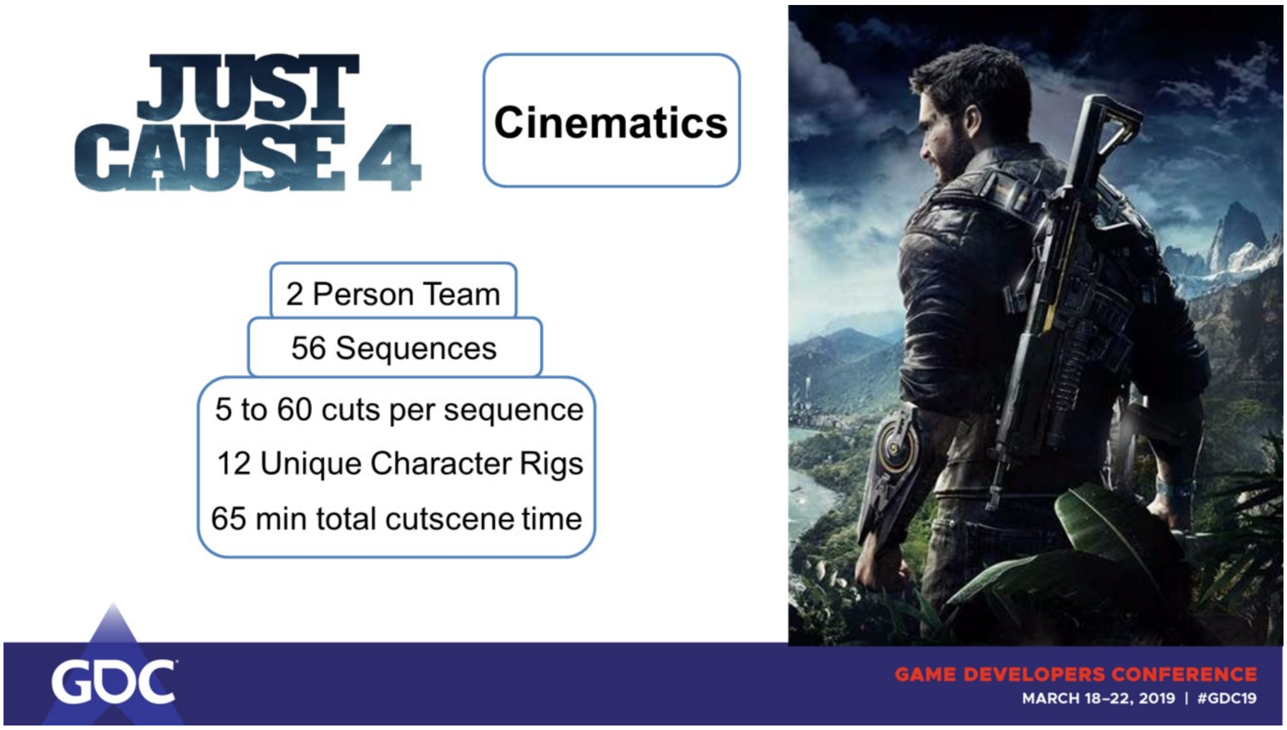
So in the end thanks to this nifty Batch tool, not only was I able to set up anything for myself really fast to start churning out lots of content through headless Maya, it opened up a ton of time saving possibilities for the animators too, and not just for exporting animations. In fact even though our pipeline called for content to be passed through Maya before export to the game, our MoBu only animator used this batch tool in conjunction with the fbx takes batch exporting tool we had for MoBu and he actually never once opened up Maya. Not once. We just had batch script set up for him to select all his fbx exports here and then the tool would go through each file, throw the animation on the applicable rig, and export that to the game. And now just a few months ago…MotionBuilder 2019 added headless mode as well…

YO DAWG, I HEARD YOU LIKE BATCHING
SO I PUT A BATCH TOOL INSIDE OF YOUR BATCH TOOL SO YOU CAN BATCH WHILE YOU BATCH
run_extreme
Options:
Select Batch Script
Export
Inverse Rotations
Reset Origin
Debug
Perforce Changelist:
Run Batch

rico_loco_run_extreme_loop_dir0_1cycles_655mps.ma
rico_loco_run_extreme_loop_dir0_1cycles_655mps_rifle_noaim.ma
rico_loco_run_extreme_loop_dir112r_2cycles_600mps_rifle_aim0.ma
rico_loco_run_extreme_loop_dir67l_2cycles_600mps_rifle_aim0.ma
rico_loco_run_extreme_stop_dir90r_rifle_aim0.ma

I plan on upgrading this baby soon and adding headless MotionBuilder into the mix.

A world where we're pushing our MotionBuilder content to/and from Maya and exporting it out to the game, all without the need of having to open up any visual/physical DCC app unless necessary is a world I certainly want to live in.


JUST CAUSE 4
GDC
GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

Cinematics

- 2 Person Team
- 56 Sequences
- 5 to 60 cuts per sequence
- 12 Unique Character Rigs
- 65 min total cutscene time

On to cinematics..

We had 2 full time team members as far as managing the actual animation and camera content goes. One of these was myself, and the other was our cinematic artist.

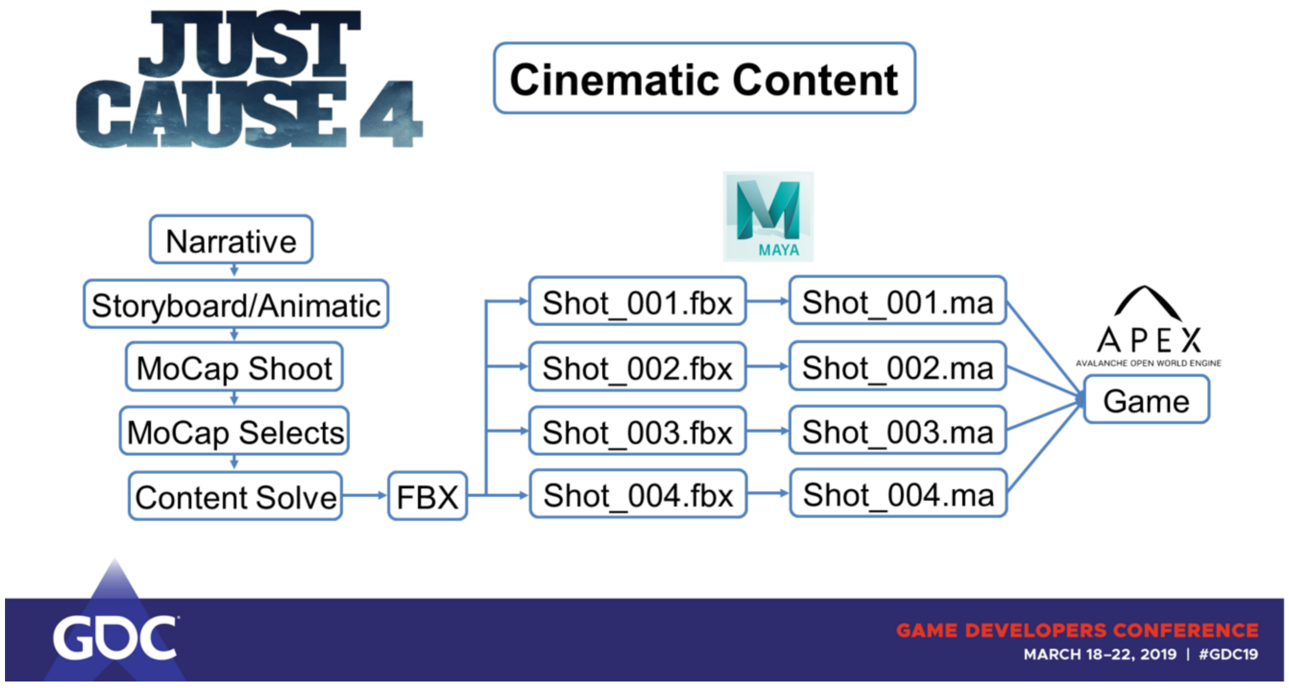
In the end we had 56 unique sequences. This number also doesn't include our single camera, single cut in-game scenes that were used to quickly set up missions.

Our smallest sequence had 5 shots that made it up, and our largest sequence had 60.

While all characters in cutscenes were bipedal humans, we had 12 unique rigs of various heights and sizes to manage and make sure were correctly used in all shots.

Between all these characters and sequences we were working with content that made up about 65 minute of actual final cutscenes

Now, actual hands on production for cutscenes didn't even start until half way through the project, and we literally didn't have actual content to work with until about 6 months before we went gold…. Wuh-oh!



Cinematic Content

Looking at the process, this may not seem too daunting when you look at what we were working with because it's fairly straightforward on the surface.

In creating JC4, like any of our AAA projects at Avalanche studios, we work with outside vendors for all Motion Capture and Content Solving when it comes to our cinematics.

To start the process, we have a narrative team at the studio as well as our cinematic artist who worked on the story and dialogue.

Which then, those ideas were transferred into storyboards and animatics to try and lock down things like pacing and camera work.

Once this was nailed down, and our actors signed on, we had a shoot. The shoot involved both our Avalanche Studios cinematics team and our vendor working together with the actors to best replicate all the factors that our animatics had tried to prove out.

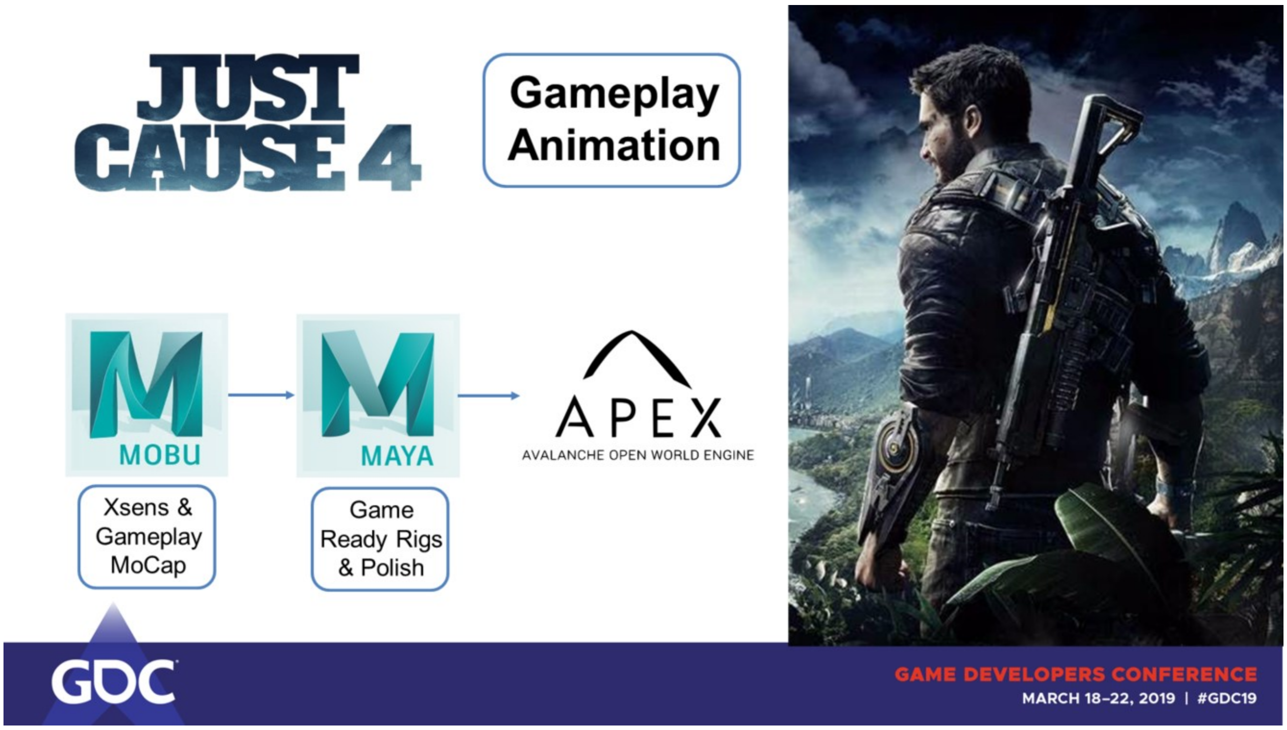
When the shoot was done, the vendor provided us with all of the takes that were shot during the shoot, and our team at the studio went through and selected which content they wanted to use and have vendor to solve for us.

After this, there was some back and forth between us and the vendor to make sure that our character rigs were set up nicely for things like good proportions based on the actor proportions and then the content was solved on our rigs, polished, and delivered to us as baked down skeletal animation in fbx file format.

Each shot was delivered as a separate FBX file with camera, characters, vehicles, weapons, and props for that shot.

That content was brought straight into Maya, transferred onto our Maya rigs.

Then exported to the game from there. Sooo we're done right? In a perfect world…yes…but making games is hardly a perfect process.



JUST CAUSE 4

Gameplay Animation

MOBU → MAYA → APEX AVALANCHE OPEN WORLD ENGINE

Xsens & Gameplay MoCap

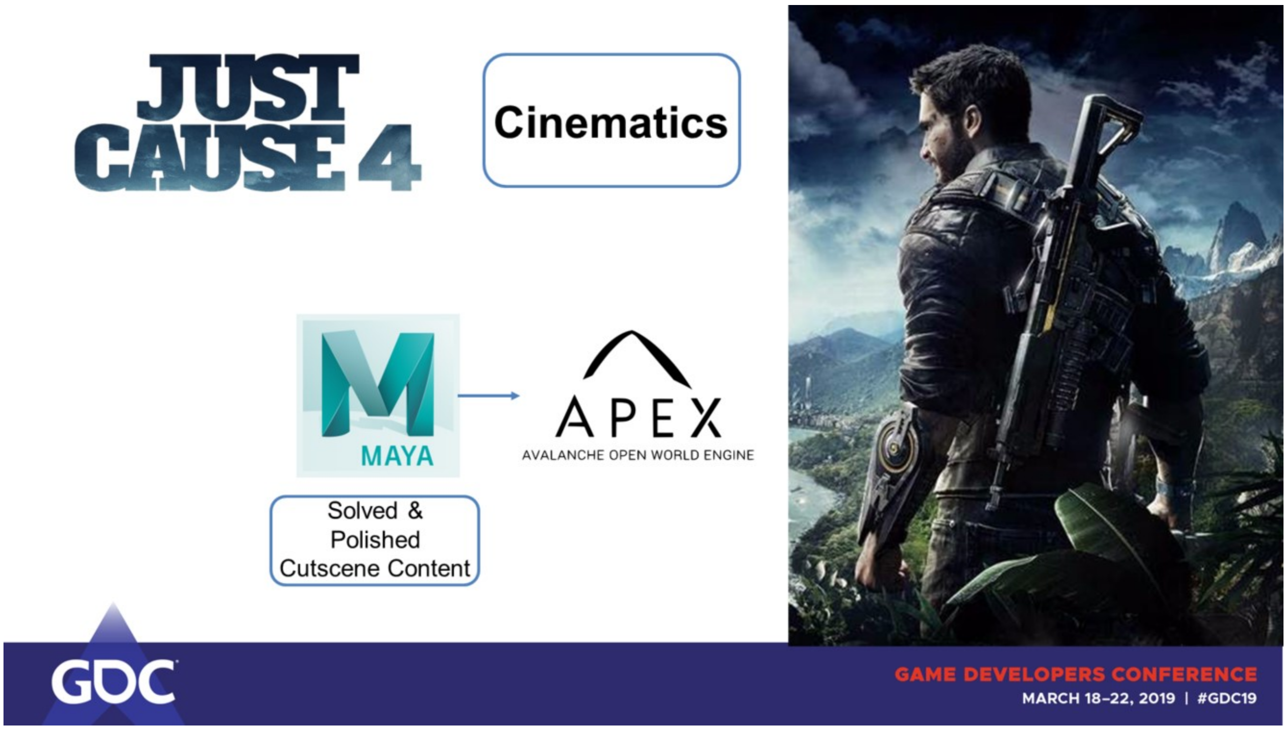
Game Ready Rigs & Polish

GDC

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

Going back to the gameplay pipeline that we were already working we had this pipeline.

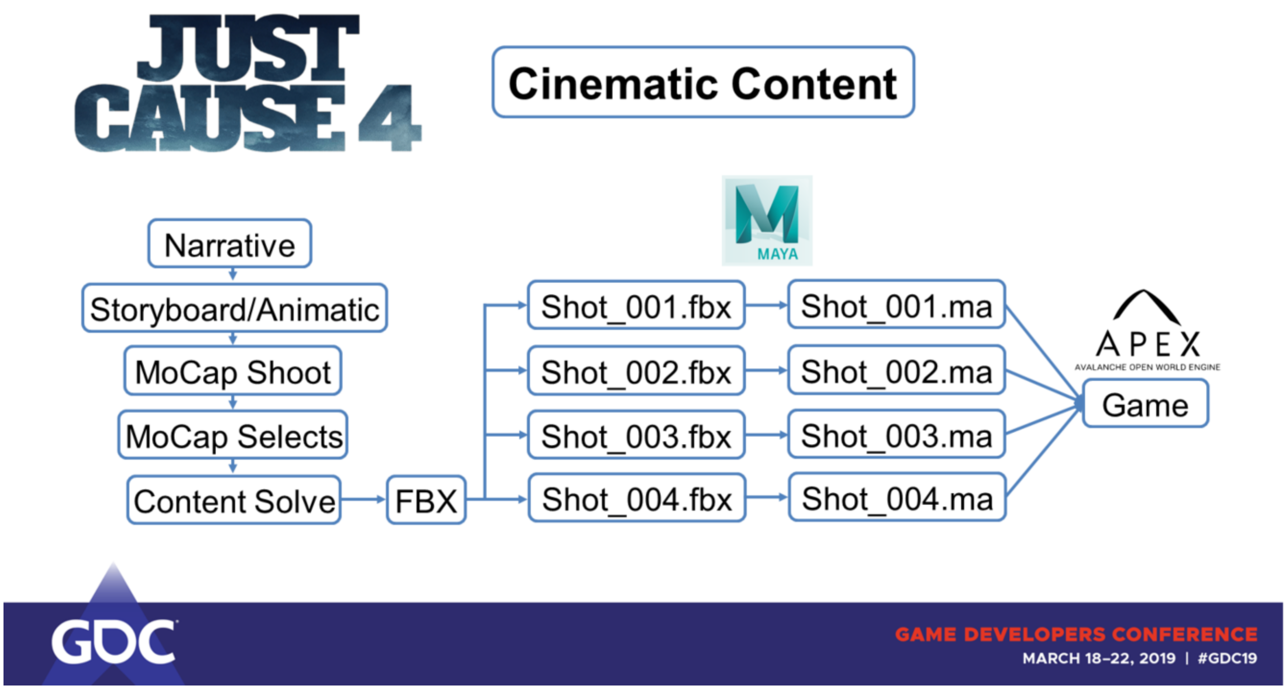
And once we started getting cinematic content from our vendor.


JUST CAUSE 4
Cinematics
MAYA
APEX
AVALANCHE OPEN WORLD ENGINE
Solved & Polished Cutscene Content
GDC
GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

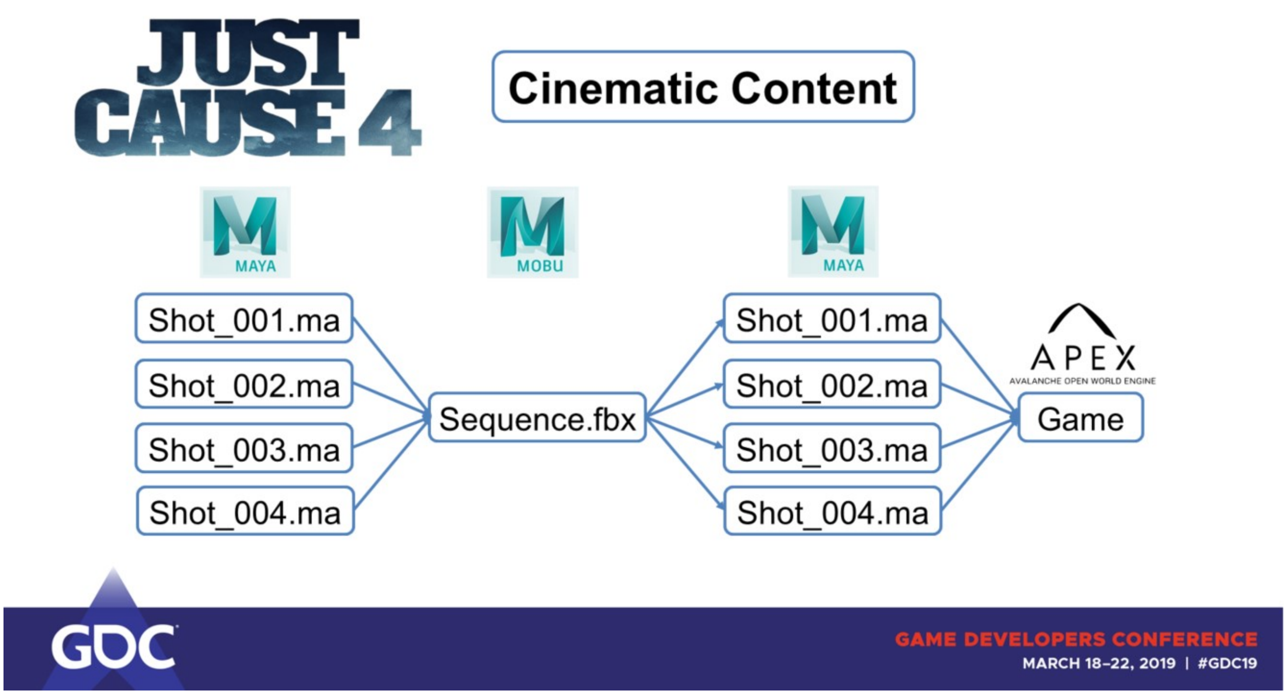
We added that content in a similar fashion to what we already did with gameplay content.

This transfer is pretty easy to do. We set up a batch script that takes all that delivered baked down skeletal MoCap data, and since the skeletons are a perfect 1:1 match with the Maya control rigs, we transfer and bake the animation over quite easily.

However, since we're making games, random emergencies or unknown needs may come from out of nowhere. Needs that with the luxury of time and a larger team can sometimes be met handedly, but time and numbers was definitely not on our side. For sake of brevity, I'm not going to go into a post-mortem right now pertaining to this issue, but I will say as a Technical Animator, my most important role is that of the problem solver. When working out any issues the first thing you do is assess the situation and come up with a viable solution based off of that.



So…going back to our cinematic content integration process.



We needed to be able to do this.

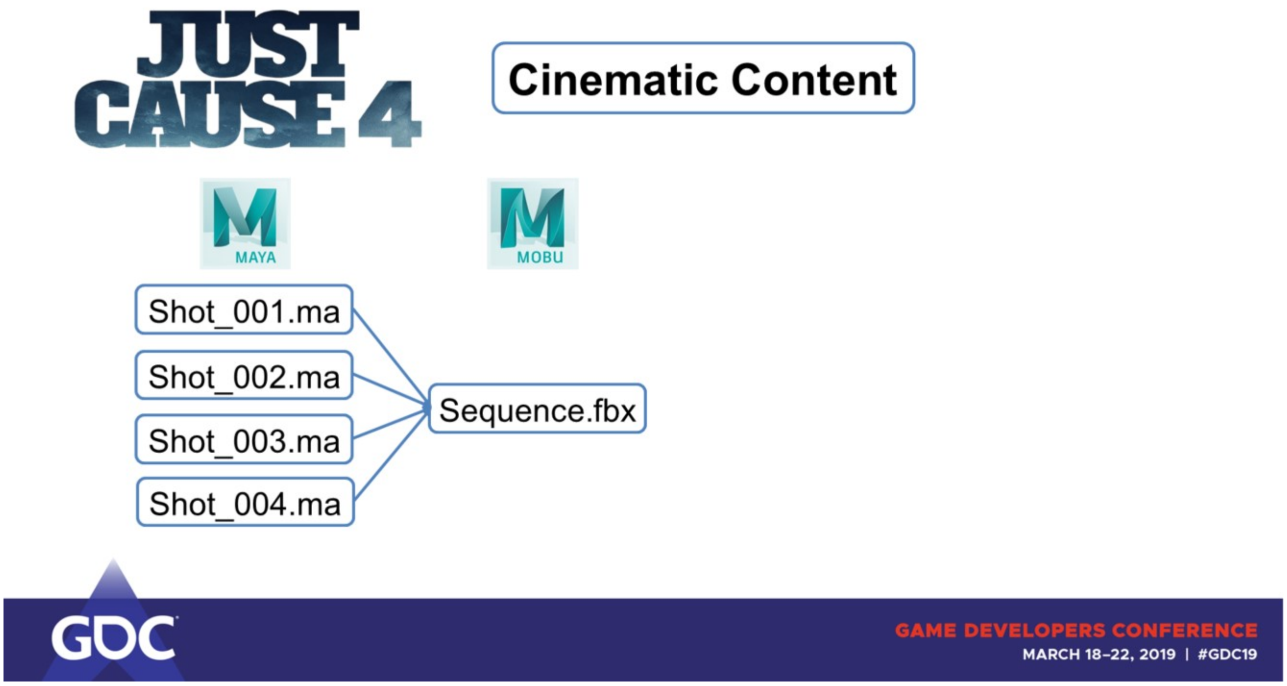
After we got our content into Maya on our game-ready rigs, we needed a way to get our cutscenes back into MoBu to allow for dealing with changes, tweaks, and critical fixes or updates.

Some of you may be asking, why would you go backwards? And I get that it isn't ideal…

But…simply put, our cinematic artist who handled all camera work, sequence edits, and any cutscene setup was 100% a motionbuilder guy. 100%. He tried Maya, he really gave it his best effort to. But the amount of time to make edits using the tools that MoBu has not to mention the great framerate that it gives you for playback (remember, animation caching in Maya JUST happened a few months ago)…well we couldn't do what we needed to without being able to take our Maya files with that solved MoCap animation from the vendor, put it back into MoBu, and then send that back out to Maya for final approval and export to the game once any new edits and changes we made.

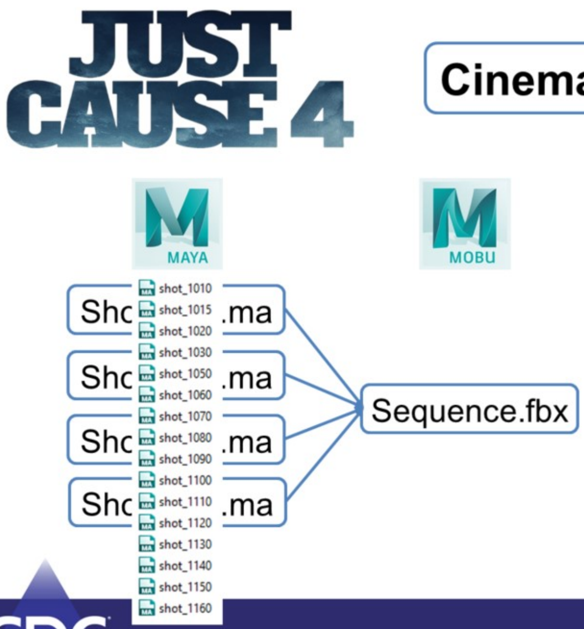
In order to best explain how we tackled our cinematics workflow, I want to break down one of our cutscenes that we had put through its paces, so let's start by watching the full cutscene. (Here I played the cutscene during the talk…it's about 2 min long)

The cutscene shows our main hero Rico, meeting with one of his supporting pals, Sargento, who gives him the part that he adds to his grapple device that allows the player to use the air lifter in game to turn anything into a flying mass. This particular cutscene has a good mix of characters, vehicles, weapons, and props, so we're managing a lot of different types of content.



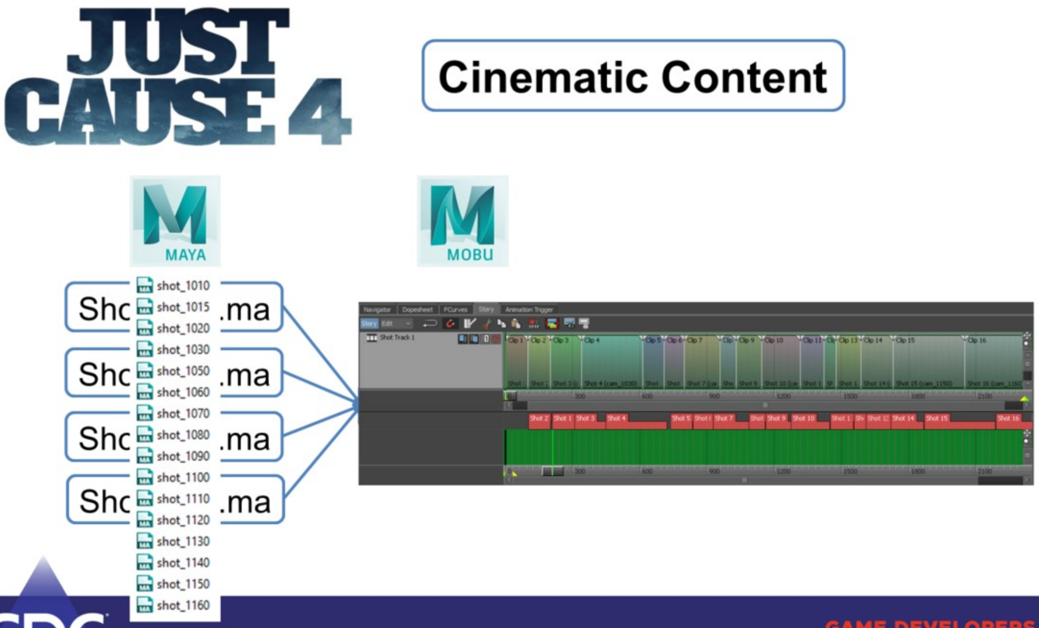
To start, let's look at the process we needed to take in order to get our Maya content into MotionBuilder.

Cinematic Content



We have our Maya files, shot by shot…but MoBu doesn’t really work this way.

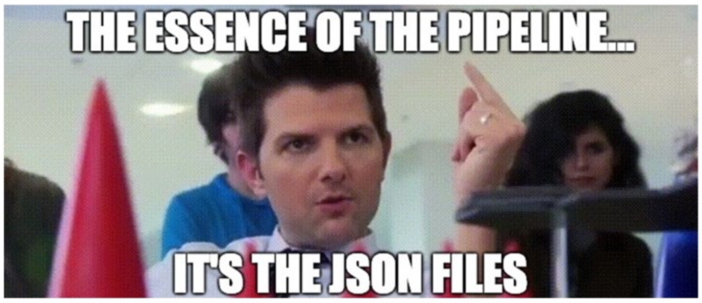
Cinematic Content



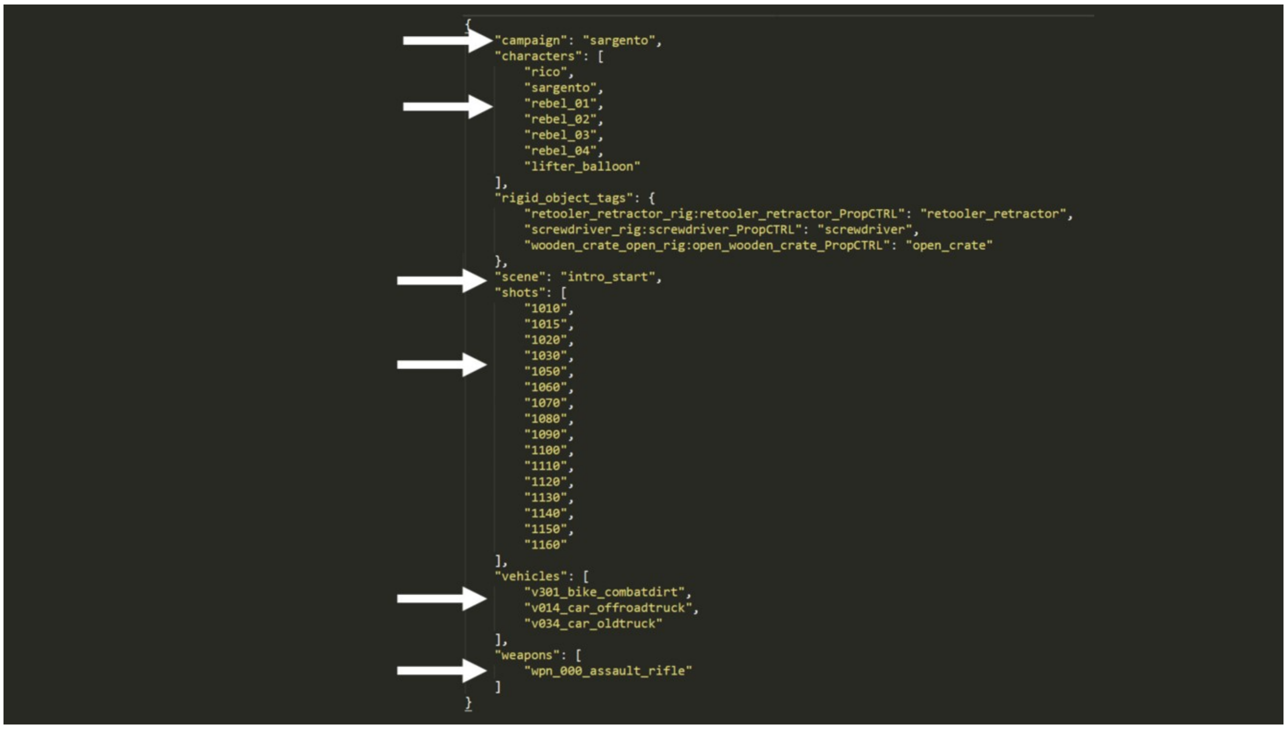
We needed to take these individual shots and transfer them into Motionbuilder in one long sequence so that they can be set up properly for editing in story mode.

JUST CAUSE 4

Cinematic Content



So, I love json. I use it all the time to store data and settings, for anything and everything I can. It’s everywhere in our pipelines probably yours too even if you don’t know it. It is the base at which the cinematics content workflow is driven.



```json
{
    "campaign": "sargento",
    "characters": [
        "rico",
        "sargento",
        "rebel_01",
        "rebel_02",
        "rebel_03",
        "rebel_04",
        "lifter_balloon"
    ],
    "rigid_object_tags": {
        "retooler_retractor_rig:retooler_retractor_PropCTRL": "retooler_retractor",
        "screwdriver_rig:screwdriver_PropCTRL": "screwdriver",
        "wooden_crate_open_rig:open_wooden_crate_PropCTRL": "open_crate"
    },
    "scene": "intro_start",
    "shots": [
        "1010",
        "1015",
        "1020",
        "1030",
        "1050",
        "1060",
        "1070",
        "1080",
        "1090",
        "1100",
        "1110",
        "1120",
        "1130",
        "1140",
        "1150",
        "1160"
    ],
    "vehicles": [
        "v301_bike_combatdirt",
        "v014_car_offroadtruck",
        "v034_car_oldtruck"
    ],
    "weapons": [
        "wpn_000_assault_rifle"
    ]
}
```

Using the power of json files to organize all of our cutscenes was the solution that not only bridged the gap between Maya and MoBu, but it organized our content in a way that is easy to parse and locate with a simple call. Every cutscene had a master json file for the cutscene data as a whole that include

The scene name.

The campaign it was part of.

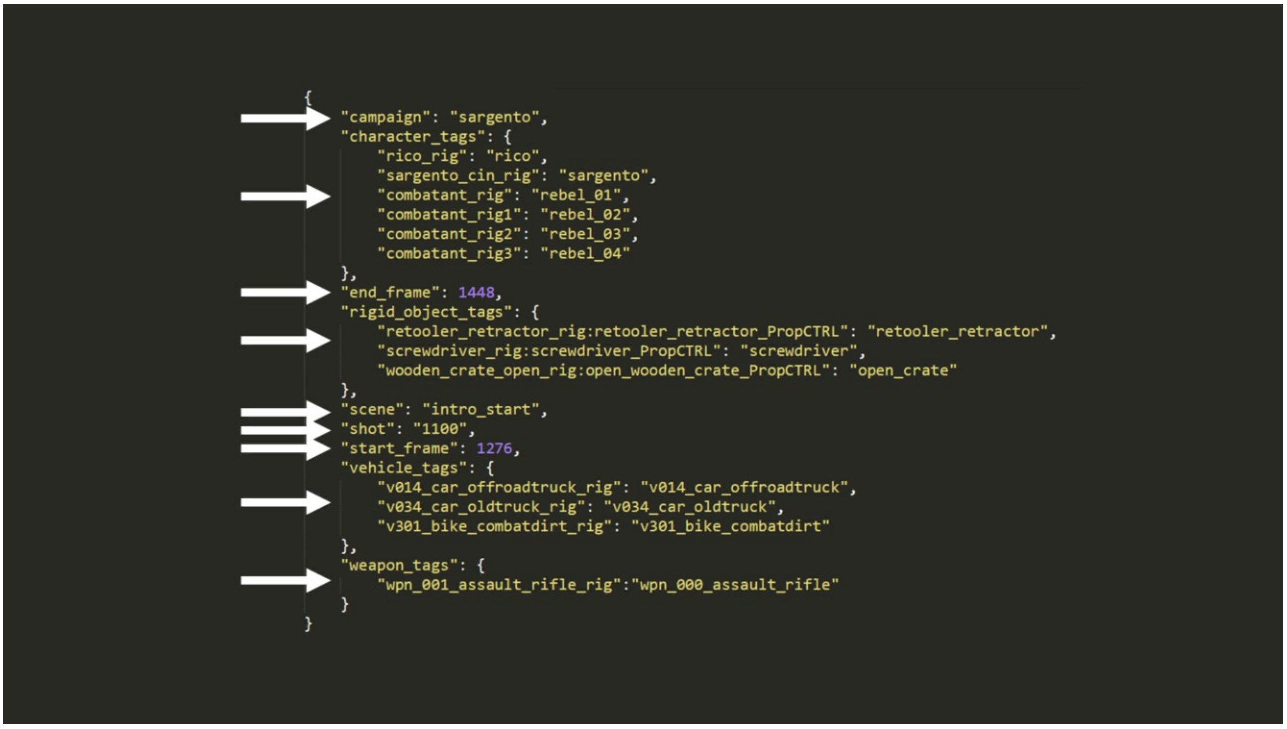
All characters.

Props.

Shots

Vehicles.

And weapons that appear in the entire cutscene at any point.



```
{
    "campaign": "sargento",
    "character_tags": {
        "rico_rig": "rico",
        "sargento_cin_rig": "sargento",
        "combatant_rig": "rebel_01",
        "combatant_rig1": "rebel_02",
        "combatant_rig2": "rebel_03",
        "combatant_rig3": "rebel_04"
    },
    "end_frame": 1448,
    "rigid_object_tags": {
        "retooler_retractor_rig:retooler_retractor_PropCTRL": "retooler_retractor",
        "screwdriver_rig:screwdriver_PropCTRL": "screwdriver",
        "wooden_crate_open_rig:open_wooden_crate_PropCTRL": "open_crate"
    },
    "scene": "intro_start",
    "shot": "1100",
    "start_frame": 1276,
    "vehicle_tags": {
        "v014_car_offroadtruck_rig": "v014_car_offroadtruck",
        "v034_car_oldtruck_rig": "v034_car_oldtruck",
        "v301_bike_combatdirt_rig": "v301_bike_combatdirt"
    },
    "weapon_tags": {
        "wpn_001_assault_rifle_rig":"wpn_000_assault_rifle"
    }
}
```

Then we had a json file for each separate shot that told us the specifics for that particular camera cut.

We again have the campaign and scene names.

The shot number.

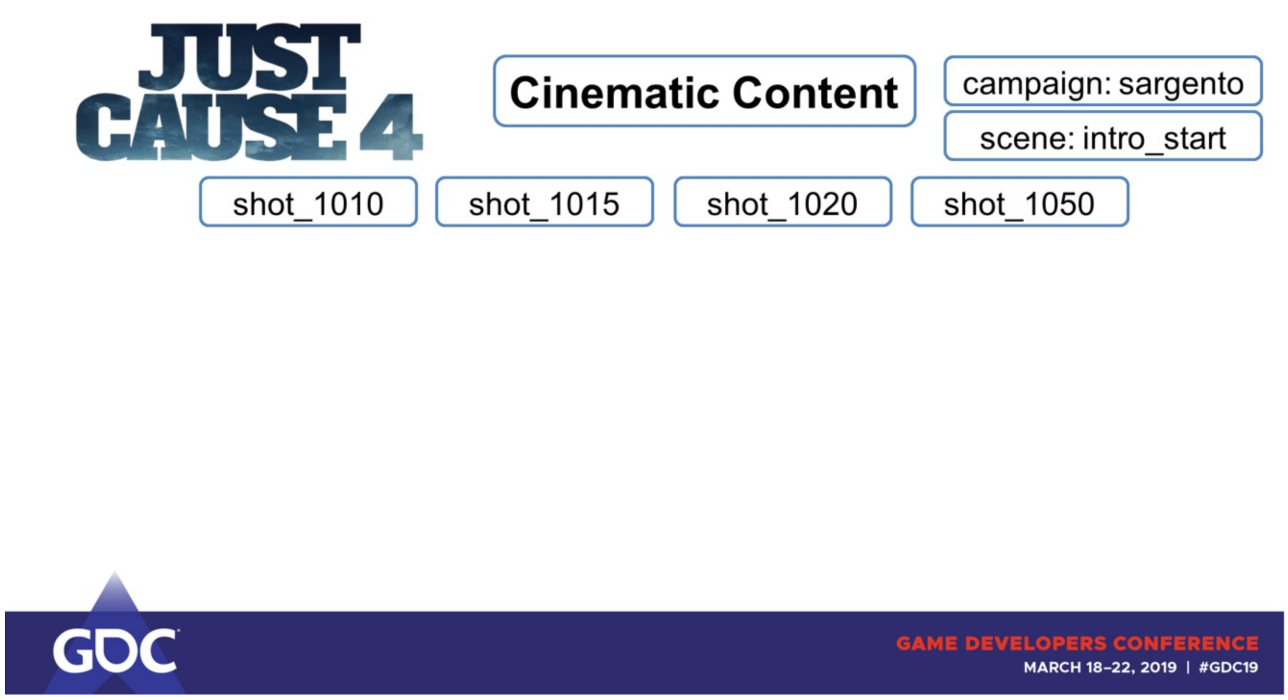
The characters that are actually in that cut.

The props in that cut.

vehicles in that cut.

And weapons in that cut.

We also have the start and end frame numbers stored.



Cinematic Content

campaign: sargento

scene: intro_start

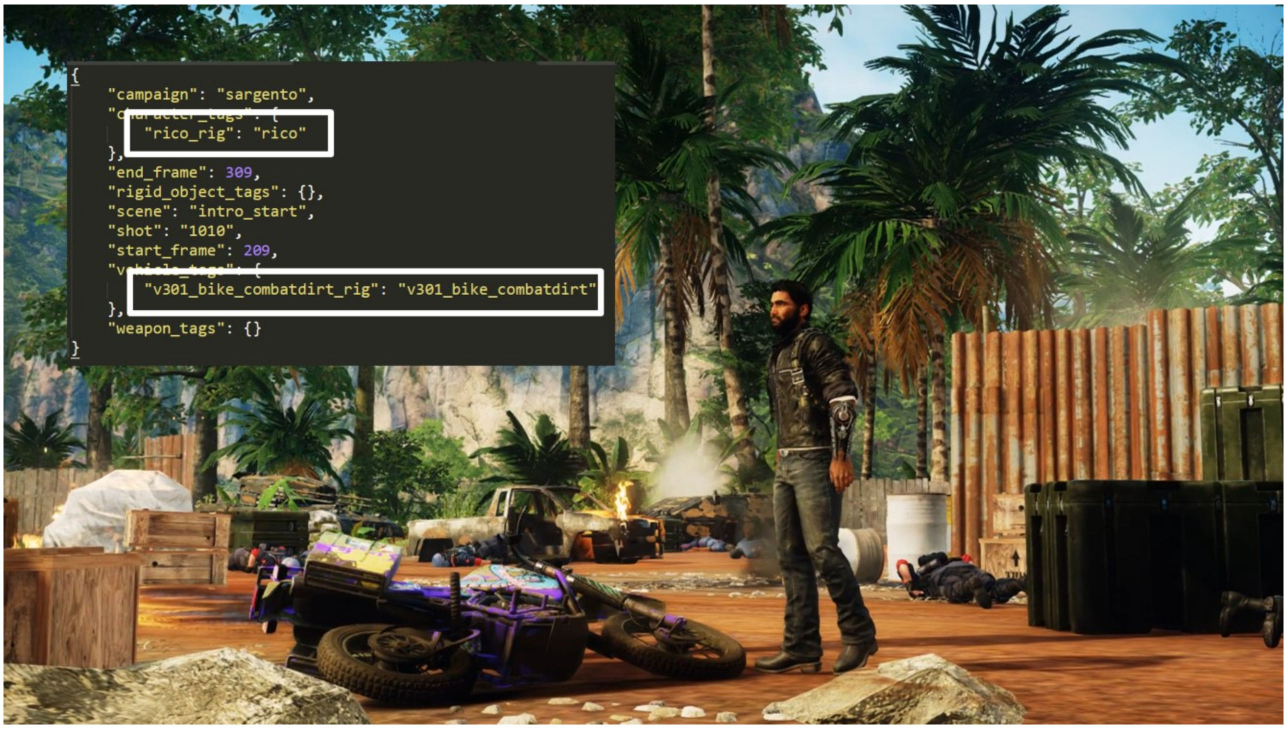
shot_1010

shot_1015

shot_1020

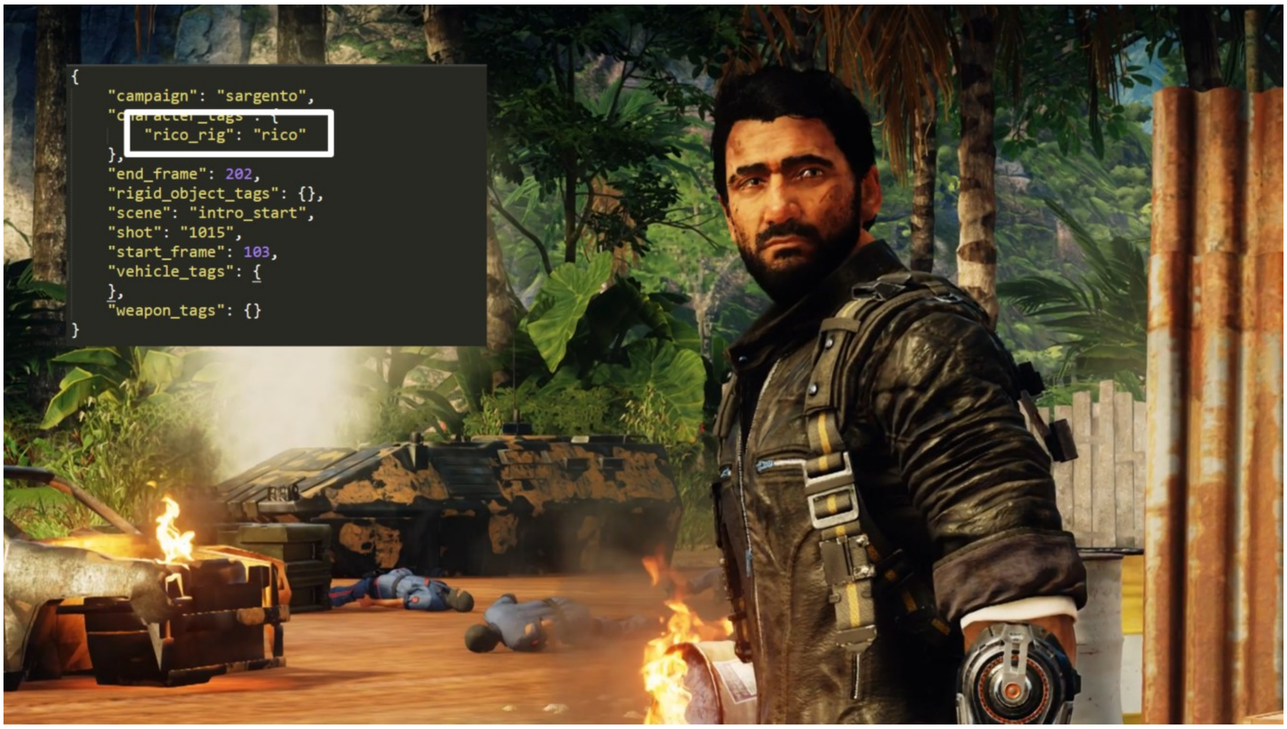
shot_1050

Just to better illustrate what this means for us, lets take a look at a few select shots within this cutscene…



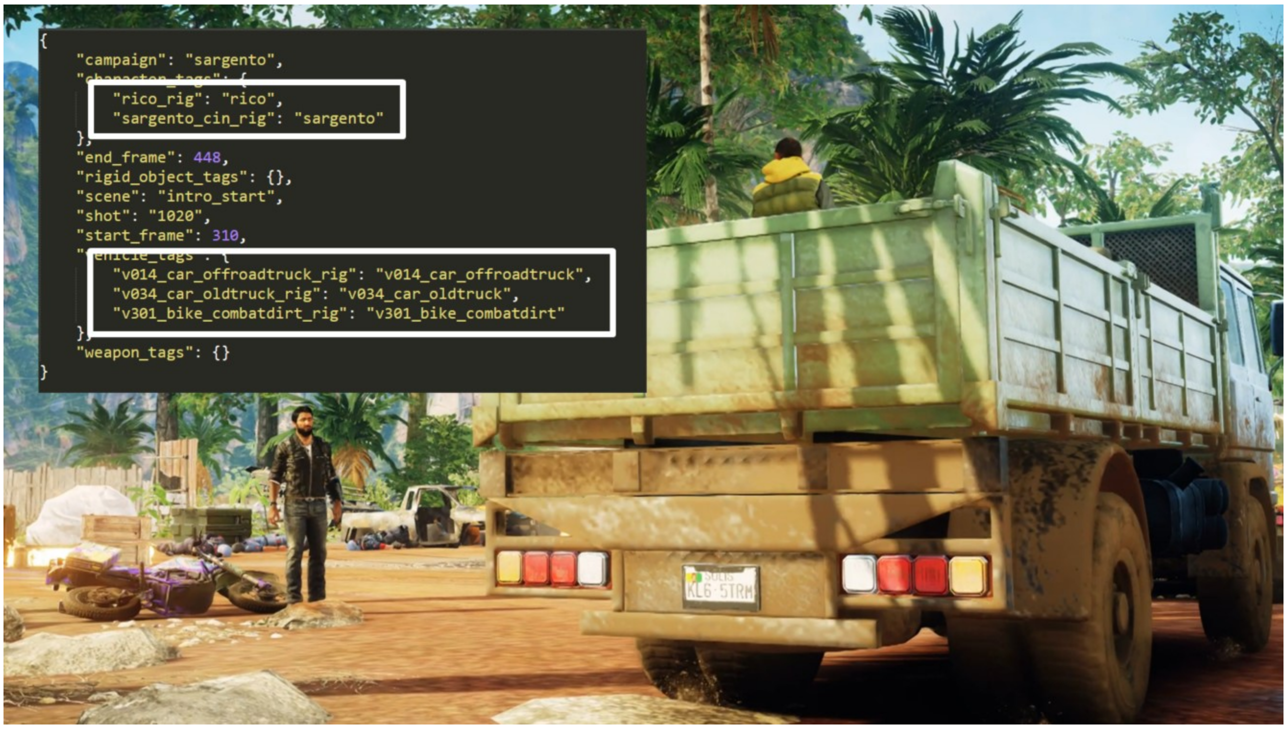
```
{
    "campaign": "sargento",
    "character_tags": {
        "rico_rig": "rico"
    },
    "end_frame": 309,
    "rigid_object_tags": {},
    "scene": "intro_start",
    "shot": "1010",
    "start_frame": 209,
    "vehicle_tags": {
        "v301_bike_combatdirt_rig": "v301_bike_combatdirt"
    },
    "weapon_tags": {}
}
```

In the first shot, we have just Rico and this dirtbike. Since there's nothing else in this scene that isn't already part of the environment, only these 2 items are stored in the json file for this particular shot.



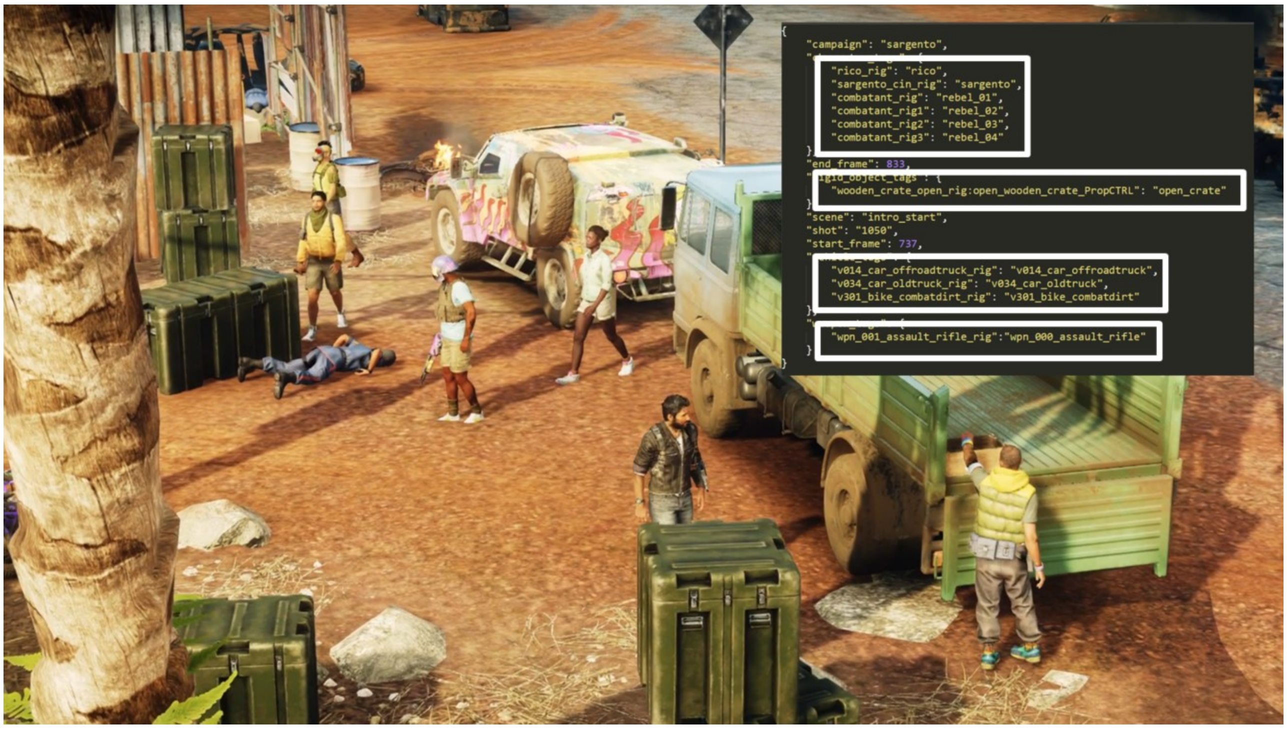
```
{
    "campaign": "sargento",
    "character_tags": {
        "rico_rig": "rico"
    },
    "end_frame": 202,
    "rigid_object_tags": {},
    "scene": "intro_start",
    "shot": "1015",
    "start_frame": 103,
    "vehicle_tags": {
    },
    "weapon_tags": {}
}
```

The second shot is only Rico, so he’s the only thing stored.



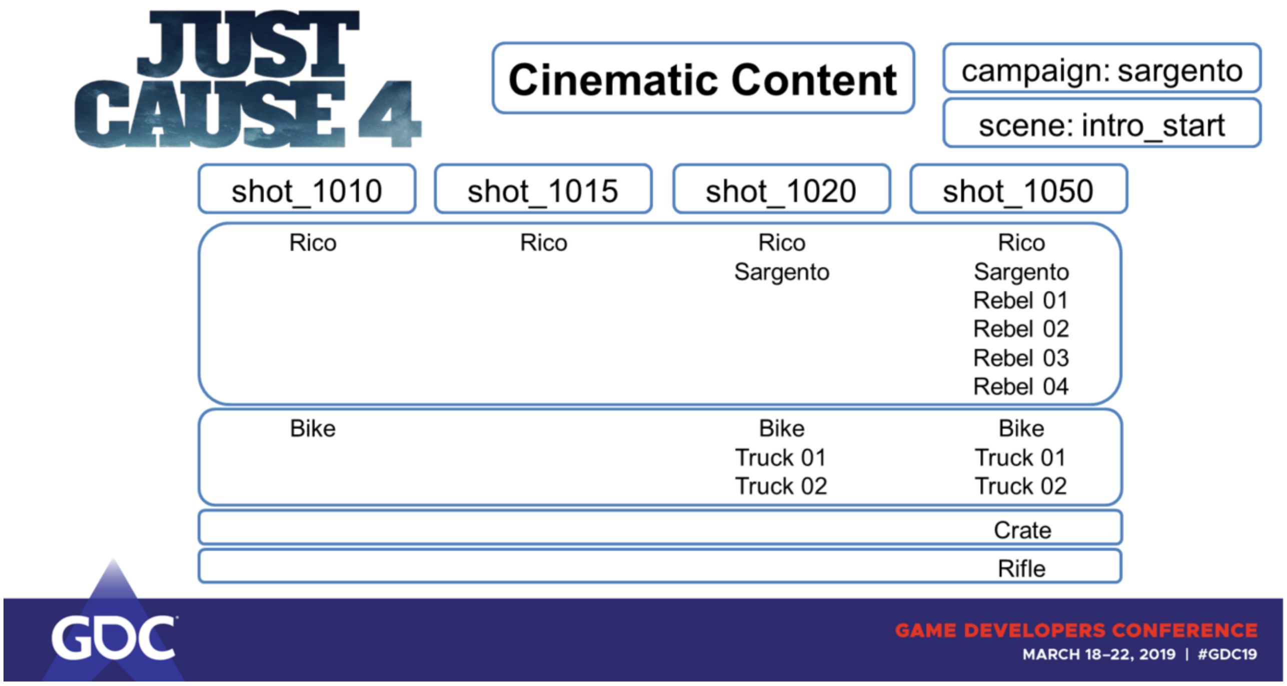
```
{
    "campaign": "sargento",
    "character_tags": {
        "rico_rig": "rico",
        "sargento_cin_rig": "sargento"
    },
    "end_frame": 448,
    "rigid_object_tags": {},
    "scene": "intro_start",
    "shot": "1020",
    "start_frame": 310,
    "vehicle_tags": {
        "v014_car_offroadtruck_rig": "v014_car_offroadtruck",
        "v034_car_oldtruck_rig": "v034_car_oldtruck",
        "v301_bike_combatdirt_rig": "v301_bike_combatdirt"
    },
    "weapon_tags": {}
}
```

The third shot sets us back to a wide angle that reintroduces the dirtbike and brings Sargento into the scene with two trucks.



```
{
    "campaign": "sargento",
    "[illegible]": {
        "rico_rig": "rico",
        "sargento_cin_rig": "sargento",
        "combatant_rig": "rebel_01",
        "combatant_rig1": "rebel_02",
        "combatant_rig2": "rebel_03",
        "combatant_rig3": "rebel_04"
    },
    "end_frame": 833,
    "rigid_object_tags": {
        "wooden_crate_open_rig:open_wooden_crate_PropCTRL": "open_crate"
    },
    "scene": "intro_start",
    "shot": "1050",
    "start_frame": 737,
    "[illegible]": {
        "v014_car_offroadtruck_rig": "v014_car_offroadtruck",
        "v034_car_oldtruck_rig": "v034_car_oldtruck",
        "v301_bike_combatdirt_rig": "v301_bike_combatdirt"
    },
    "[illegible]": {
        "wpn_001_assault_rifle_rig":"wpn_000_assault_rifle"
    }
}
```

I'm skipping over the fourth shot to the fifth one since this shot includes a little bit of everything. All 4 characters are seen, as are all 3 vehicles…the dirtbike is actually hiding on the left behind the tree, I promise, it includes a prop..which is the crate in the back of the truck, and if you can't tell, the rebel there in the middle is holding a weapon.



Cinematic Content

campaign: sargento

scene: intro_start

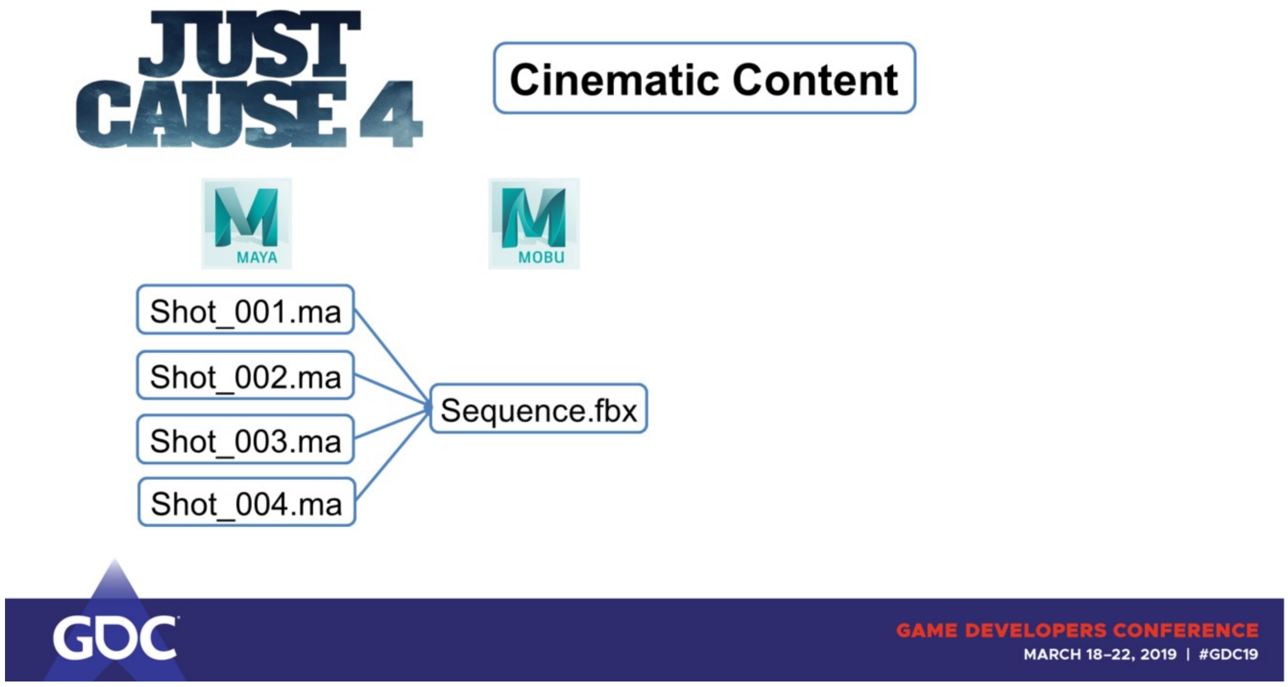
shot_1010	shot_1015	shot_1020	shot_1050
Rico	Rico	Rico Sargento	Rico Sargento Rebel 01 Rebel 02 Rebel 03 Rebel 04
Bike		Bike Truck 01 Truck 02	Bike Truck 01 Truck 02
			Crate
			Rifle

In an effort to not completely force you to look at code, here’s what this looks like in a more visually pleasing way.

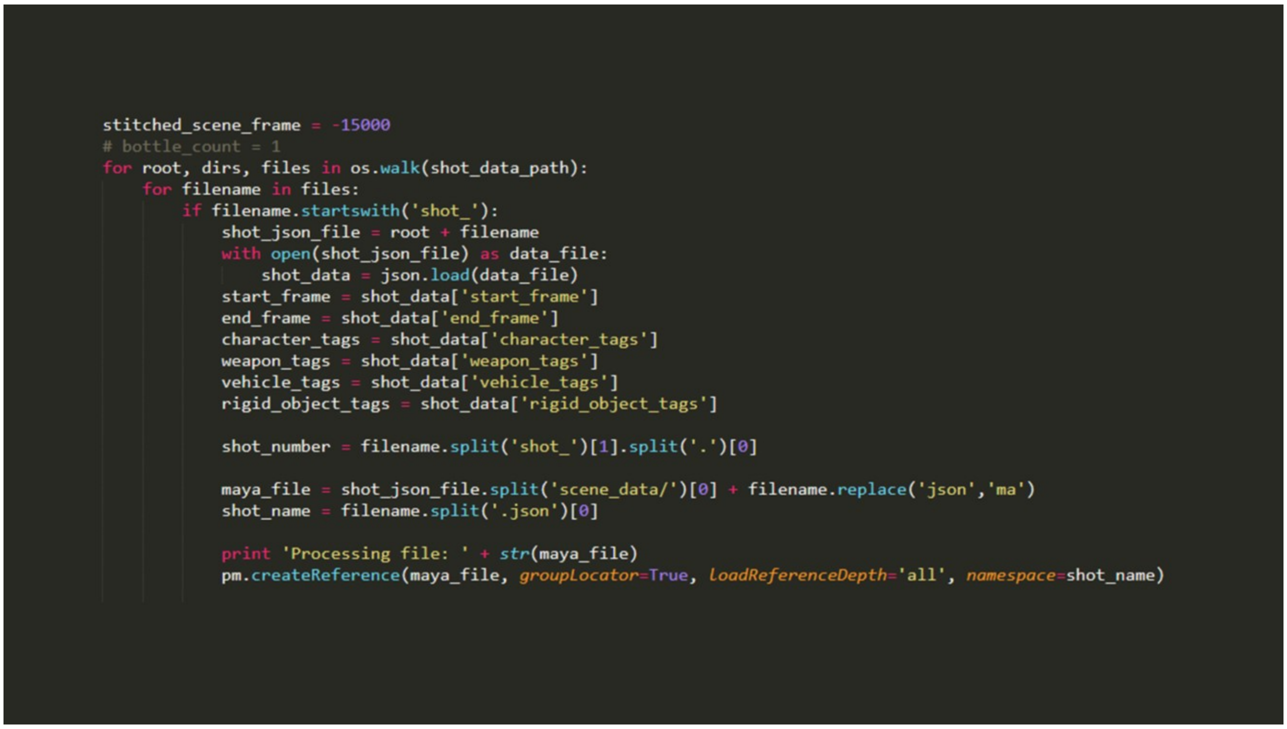
Now, because we have all this finer detail stored the way we do, I don’t have to open up Maya to actually find out what is in any given shot, which I’ll say is far easier, quicker, and nicer than trying to find this stuff on a spreedsheet.

I also can run a cutscene export headless, in our sweet batch tool and filter it down to only export Sargento because maybe we made a small rig update or added a secondary animation pass on his jacket. The batch process that we set up will parse every json file for the scene, and instead of wasting time opening up shots where Sargento doesn’t exist to check that he’s there, it will immediately go straight for only the files that he’s in, passing over those that he is not. For a small team like ours, this time savings when you think about the number of characters, scenes, and shots that we were working with was a huuuuge huge win.

This also made it easier to shot by shot, deal with our Maya to MoBu transfer.

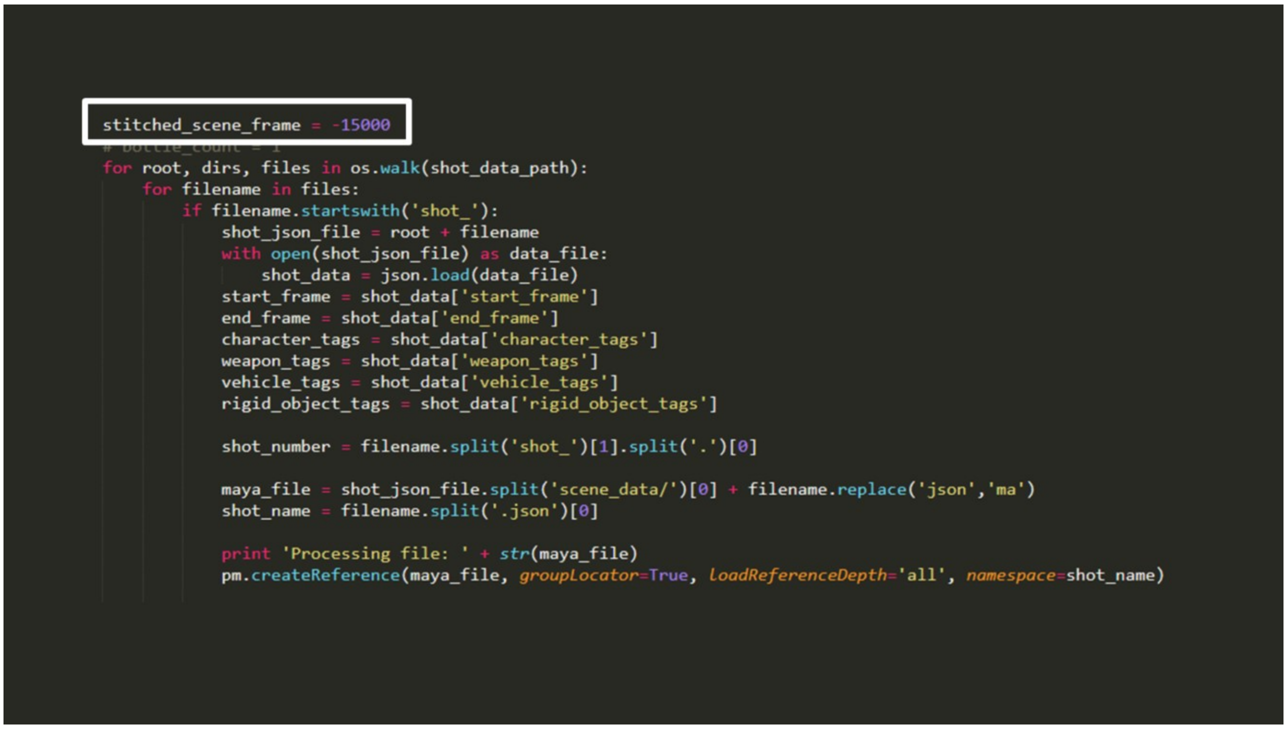


Let’s start our content passing process on the Maya side. It’s time to take our entire sequence that is cut up into multiple files in Maya and stitch all that together into one giant sequence of content so that it is “MoBu Friendly” for our cinematic artist to use with story mode and the camera switcher.


```
stitched_scene_frame = -15000
# bottle_count = 1
for root, dirs, files in os.walk(shot_data_path):
    for filename in files:
        if filename.startswith('shot_'):
            shot_json_file = root + filename
            with open(shot_json_file) as data_file:
                shot_data = json.load(data_file)
            start_frame = shot_data['start_frame']
            end_frame = shot_data['end_frame']
            character_tags = shot_data['character_tags']
            weapon_tags = shot_data['weapon_tags']
            vehicle_tags = shot_data['vehicle_tags']
            rigid_object_tags = shot_data['rigid_object_tags']

            shot_number = filename.split('shot_')[1].split('.')[0]

            maya_file = shot_json_file.split('scene_data/')[0] + filename.replace('json','ma')
            shot_name = filename.split('.json')[0]

            print 'Processing file: ' + str(maya_file)
            pm.createReference(maya_file, groupLocator=True, loadReferenceDepth='all', namespace=shot_name)
```

A short look into part of the script that does this for us.

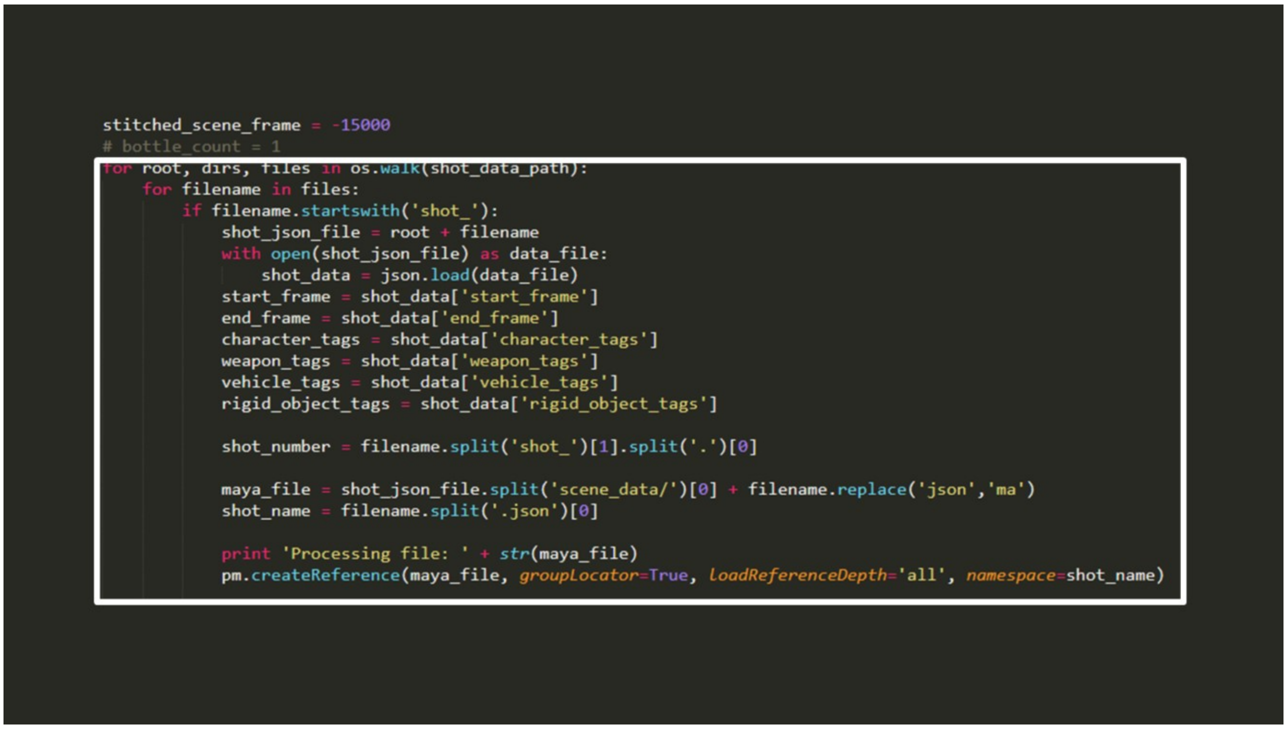
When we run our process, we open a fresh maya scene, let’s call this the master file, and…



```
stitched_scene_frame = -15000
# bottle_count = 1
for root, dirs, files in os.walk(shot_data_path):
    for filename in files:
        if filename.startswith('shot_'):
            shot_json_file = root + filename
            with open(shot_json_file) as data_file:
                shot_data = json.load(data_file)
            start_frame = shot_data['start_frame']
            end_frame = shot_data['end_frame']
            character_tags = shot_data['character_tags']
            weapon_tags = shot_data['weapon_tags']
            vehicle_tags = shot_data['vehicle_tags']
            rigid_object_tags = shot_data['rigid_object_tags']

            shot_number = filename.split('shot_')[1].split('.')[0]

            maya_file = shot_json_file.split('scene_data/')[0] + filename.replace('json','ma')
            shot_name = filename.split('.json')[0]

            print 'Processing file: ' + str(maya_file)
            pm.createReference(maya_file, groupLocator=True, loadReferenceDepth='all', namespace=shot_name)
```

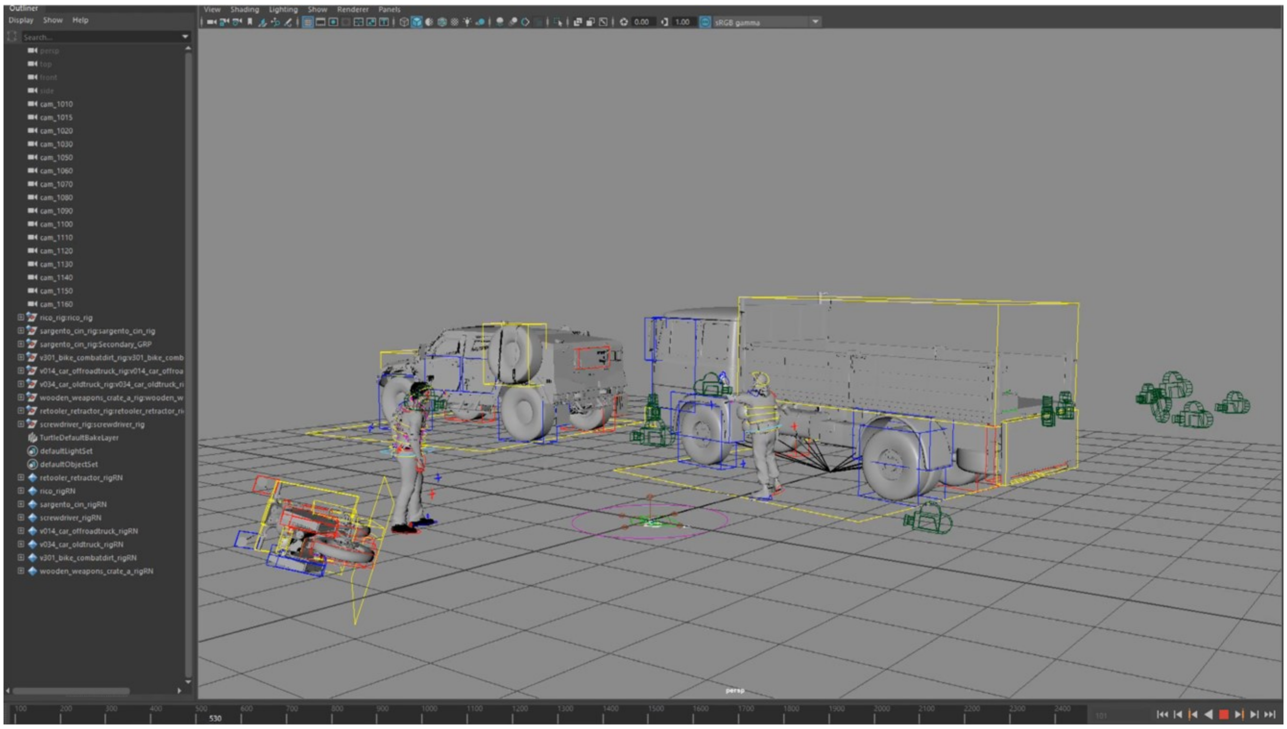
…immediately start at frame -15,000. This is to ensure that as each individual shot file is having it's content transferred over, we don't get clashing or overlapping keyframes that screw things up.

Before this actually, the json file for the whole cutscene itself is read and all cameras, characters, etc that exist at any point in the cutscene are referenced into the Maya scene.



```
stitched_scene_frame = -15000
# bottle_count = 1
for root, dirs, files in os.walk(shot_data_path):
    for filename in files:
        if filename.startswith('shot_'):
            shot_json_file = root + filename
            with open(shot_json_file) as data_file:
                shot_data = json.load(data_file)
            start_frame = shot_data['start_frame']
            end_frame = shot_data['end_frame']
            character_tags = shot_data['character_tags']
            weapon_tags = shot_data['weapon_tags']
            vehicle_tags = shot_data['vehicle_tags']
            rigid_object_tags = shot_data['rigid_object_tags']

            shot_number = filename.split('shot_')[1].split('.')[0]

            maya_file = shot_json_file.split('scene_data/')[0] + filename.replace('json','ma')
            shot_name = filename.split('.json')[0]

            print 'Processing file: ' + str(maya_file)
            pm.createReference(maya_file, groupLocator=True, loadReferenceDepth='all', namespace=shot_name)
```

Since everything we need exists in the master Maya file, we loop through all the individual json file shots, referencing in the corresponding Maya shot file for each as we do and all relevant content for each shot gets baked down to the rigs in our master file.

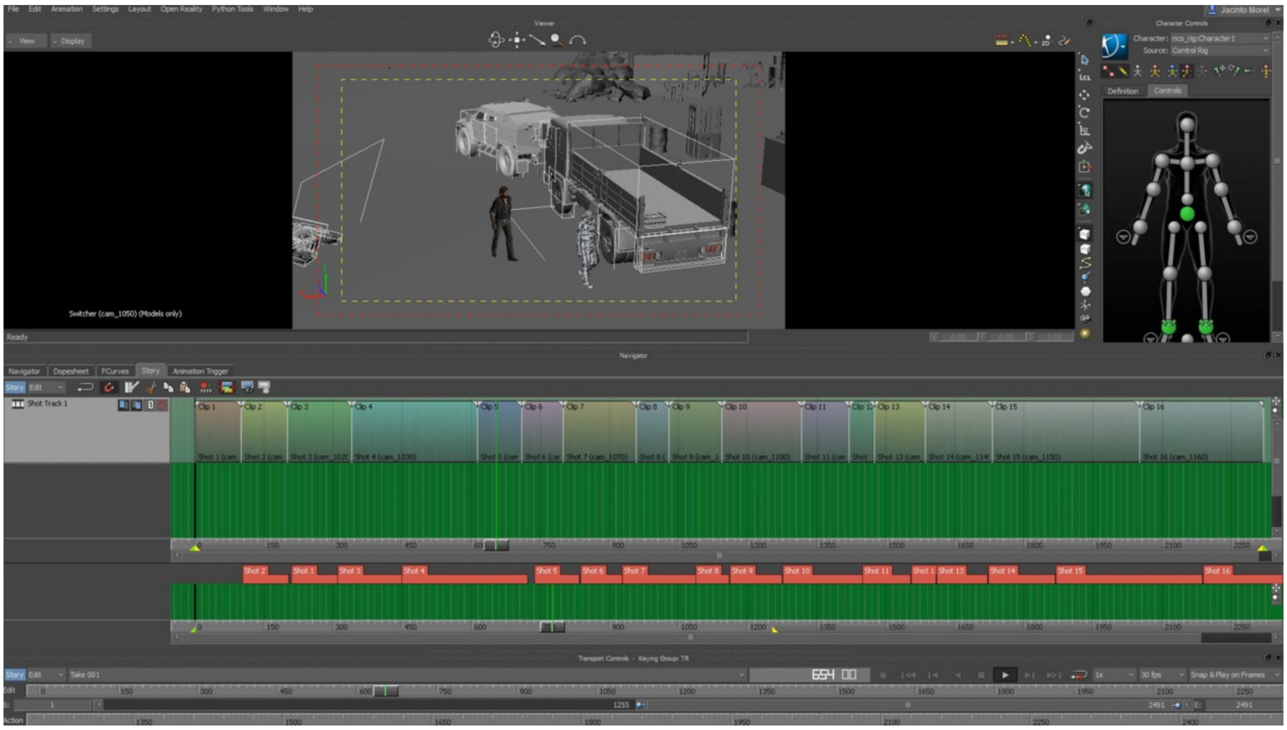


As each shot is transferred, the animation is baked down from the start and end frames of that single shot, then all keyframes are grabbed and moved in place behind the last shots end frame. By the time the last shot is done transferring we have one long sequence with each shot having been neatly placed one after another.

Once all shots are fully transferred into this master maya file, all keyframes are grabbed and moved up to frame 101 so we're not dealing with crazy negative space on the timeline. You could easily just make 0 your start frame but I personally like to leave frames 0-100 open in case of any necessary padding that may be needed, but I digress.

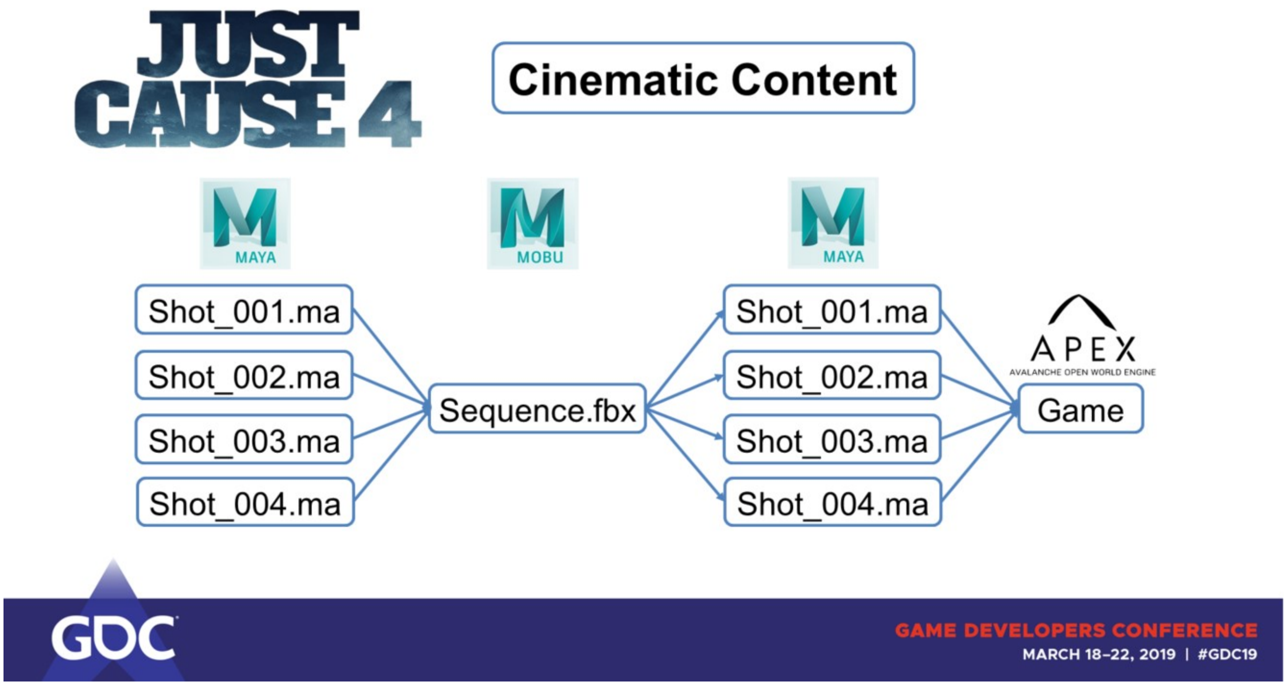
As you can see we have all rigs and cameras here nice and neatly spliced into one. There's another script that bakes alllll this animation down then cleans up and deletes everything but the models that are skinned to these baked down skeletons that are also characterized, because remember way back when in this very presentation I showed you that regardless of the type of rig, all our rigs are characterized.

So now the file is exported to FBX and is ready for some story editing in MoBu.



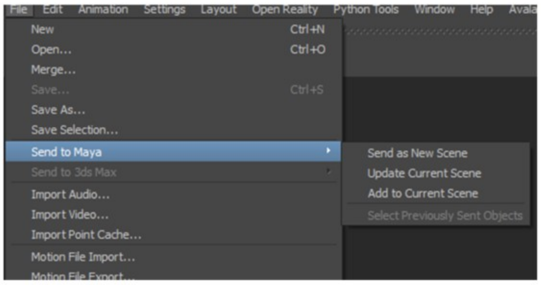
So picture this…The maya file you just saw that was baked down and transferred into MoBu has made it's way to the cinematic artist and this person has made all the necessary edits to cameras, character positions, whatever else may have needed to be fixed. This person saves this file and says "Hello Brian, it is ready for you."

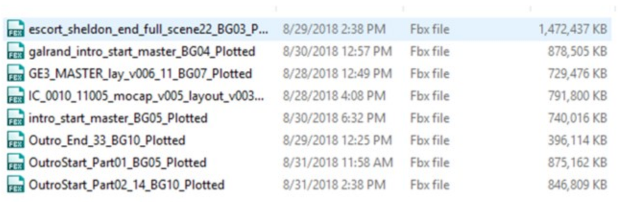
So here's that MoBu file all set up and ready to go. Since we're ready to send back to Maya we need to split up every one of these individual clips that exist in story….



…and then export them out into individual Maya files as per our pipeline in order for us to finish off the scene and get it in game.

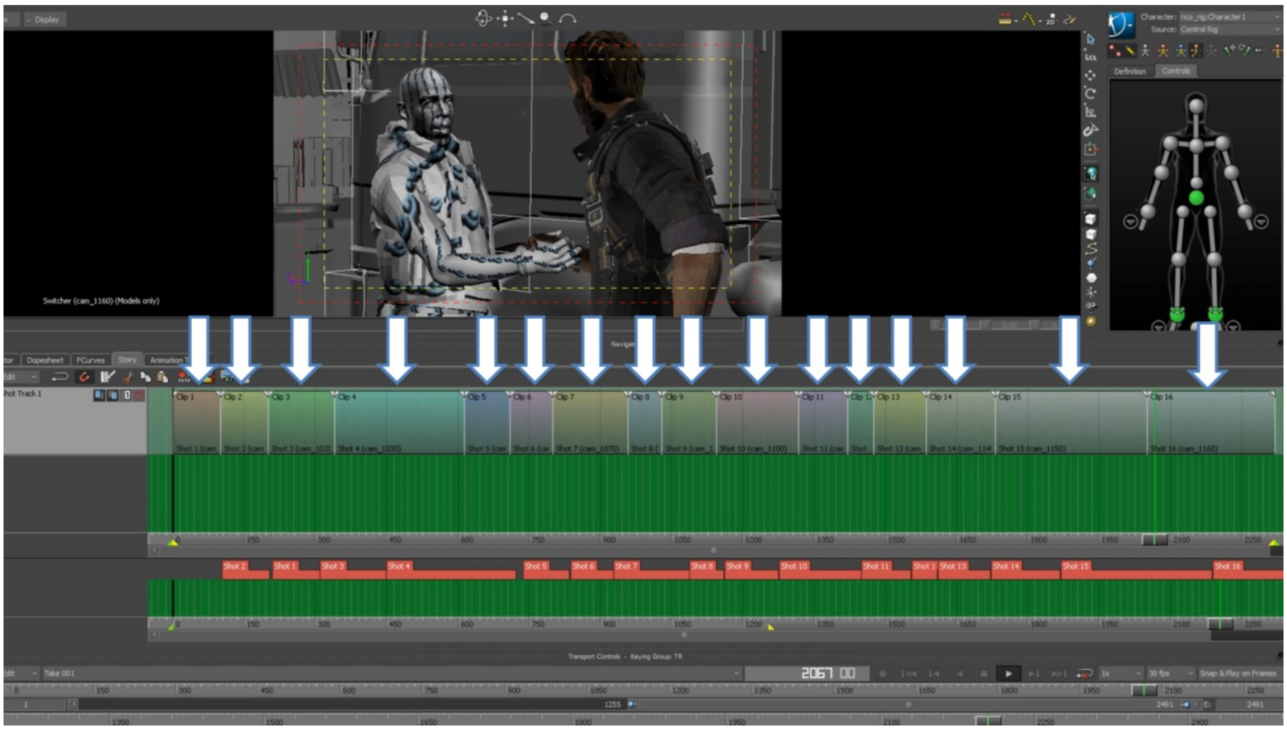
Cinematic Content



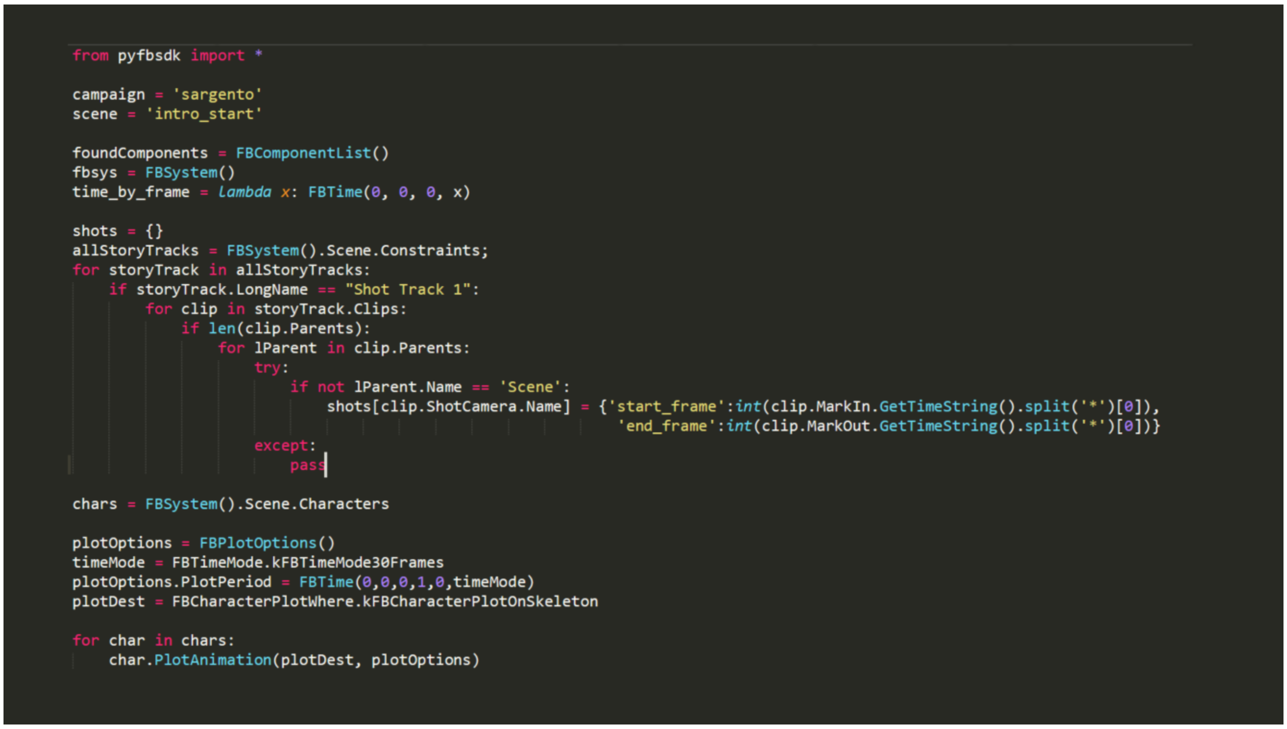


escort_sheldon_end_full_scene22_BG03_P...	8/29/2018 2:38 PM	Fbx file	1,472,437 KB
galrand_intro_start_master_BG04_Plotted	8/30/2018 12:57 PM	Fbx file	878,505 KB
GE3_MASTER_lay_v006_11_BG07_Plotted	8/28/2018 12:49 PM	Fbx file	729,476 KB
IC_0010_11005_mocap_v005_layout_v003...	8/28/2018 4:08 PM	Fbx file	791,800 KB
intro_start_master_BG05_Plotted	8/30/2018 6:32 PM	Fbx file	740,016 KB
Outro_End_33_BG10_Plotted	8/29/2018 12:25 PM	Fbx file	396,114 KB
OutroStart_Part01_BG05_Plotted	8/31/2018 11:58 AM	Fbx file	875,162 KB
OutroStart_Part02_14_BG10_Plotted	8/31/2018 2:38 PM	Fbx file	846,809 KB

In order to do this, we need to run some commands in MoBu that export out data that can be brought into Maya QUICKLY and with little to no load times. Using the "Send to Maya" command or opening an FBX straight up in MotionBuilder means that you send EVERYTHING all at once into Maya, and it's not necessarily the cleanest result…so we don't want that. File sizes get HUGE (as evidenced by the FBX scenes here on the right) and these can take forever to even just open up in Maya. To avoid these long and arduous transfer times, we break down the MoBu scenes for transfer, then fit them back together in Maya.

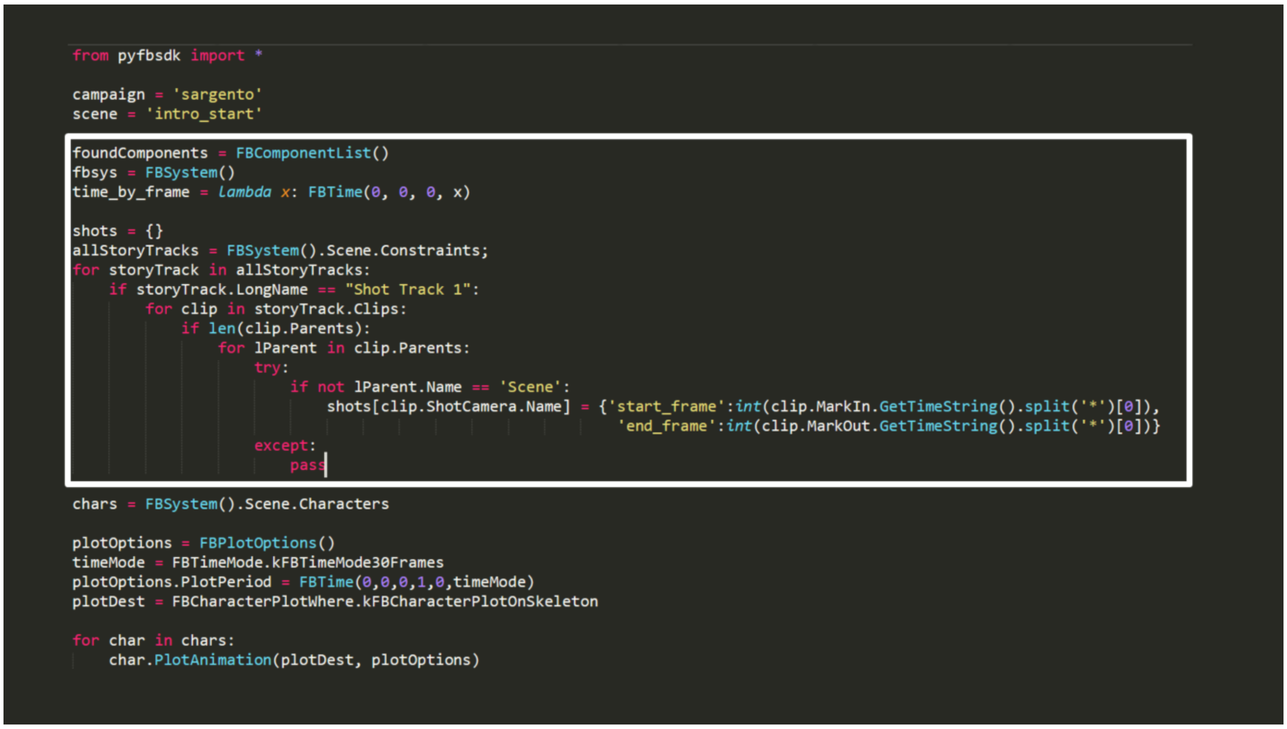


Not only do we have to break each of these clips down and export them separately, but we need to make sure we do so in a manner that doesn’t export out large file sizes.



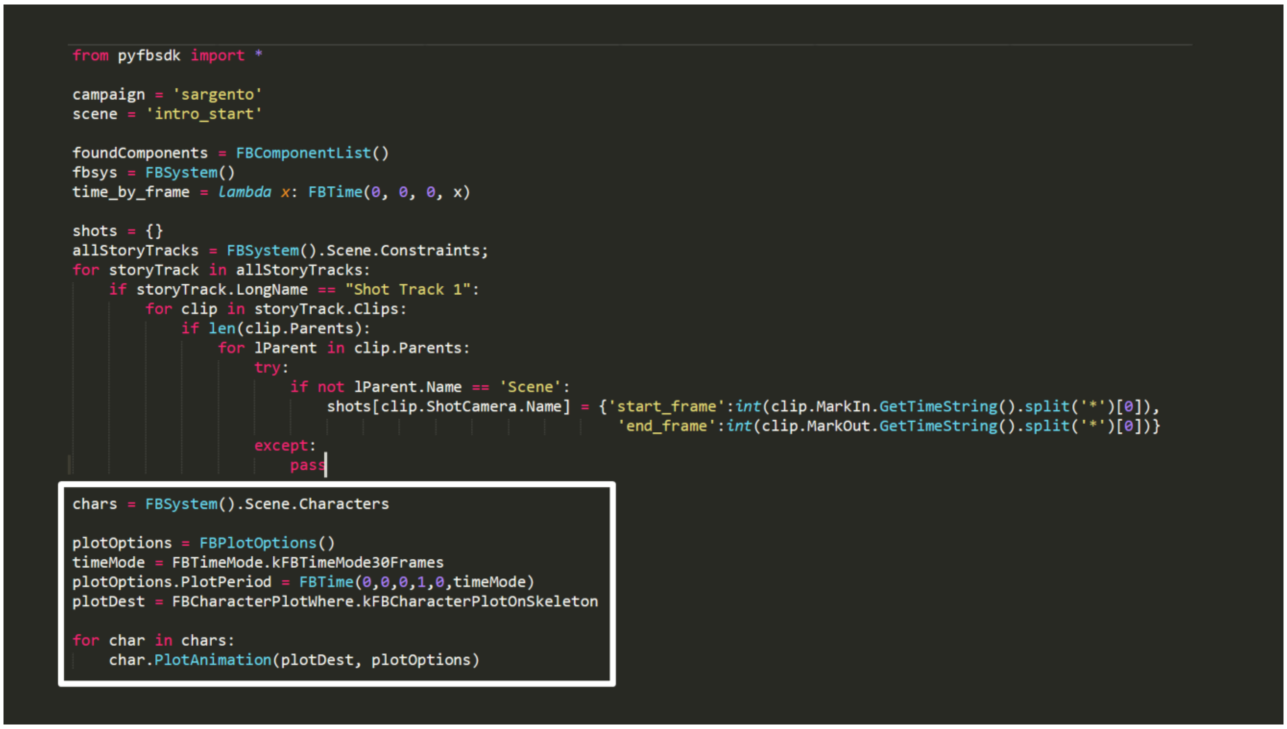
```python
from pyfbsdk import *

campaign = 'sargento'
scene = 'intro_start'

foundComponents = FBComponentList()
fbsys = FBSystem()
time_by_frame = lambda x: FBTime(0, 0, 0, x)

shots = {}
allStoryTracks = FBSystem().Scene.Constraints;
for storyTrack in allStoryTracks:
    if storyTrack.LongName == "Shot Track 1":
        for clip in storyTrack.Clips:
            if len(clip.Parents):
                for lParent in clip.Parents:
                    try:
                        if not lParent.Name == 'Scene':
                            shots[clip.ShotCamera.Name] = {'start_frame':int(clip.MarkIn.GetTimeString().split('*')[0]),
                                                            'end_frame':int(clip.MarkOut.GetTimeString().split('*')[0])}
                    except:
                        pass

chars = FBSystem().Scene.Characters

plotOptions = FBPlotOptions()
timeMode = FBTimeMode.kFBTimeMode30Frames
plotOptions.PlotPeriod = FBTime(0,0,0,1,0,timeMode)
plotDest = FBCharacterPlotWhere.kFBCharacterPlotOnSkeleton

for char in chars:
    char.PlotAnimation(plotDest, plotOptions)
```

Since we’re getting close to the end here, I’ll spare you long code explanations and briefly run through this process.



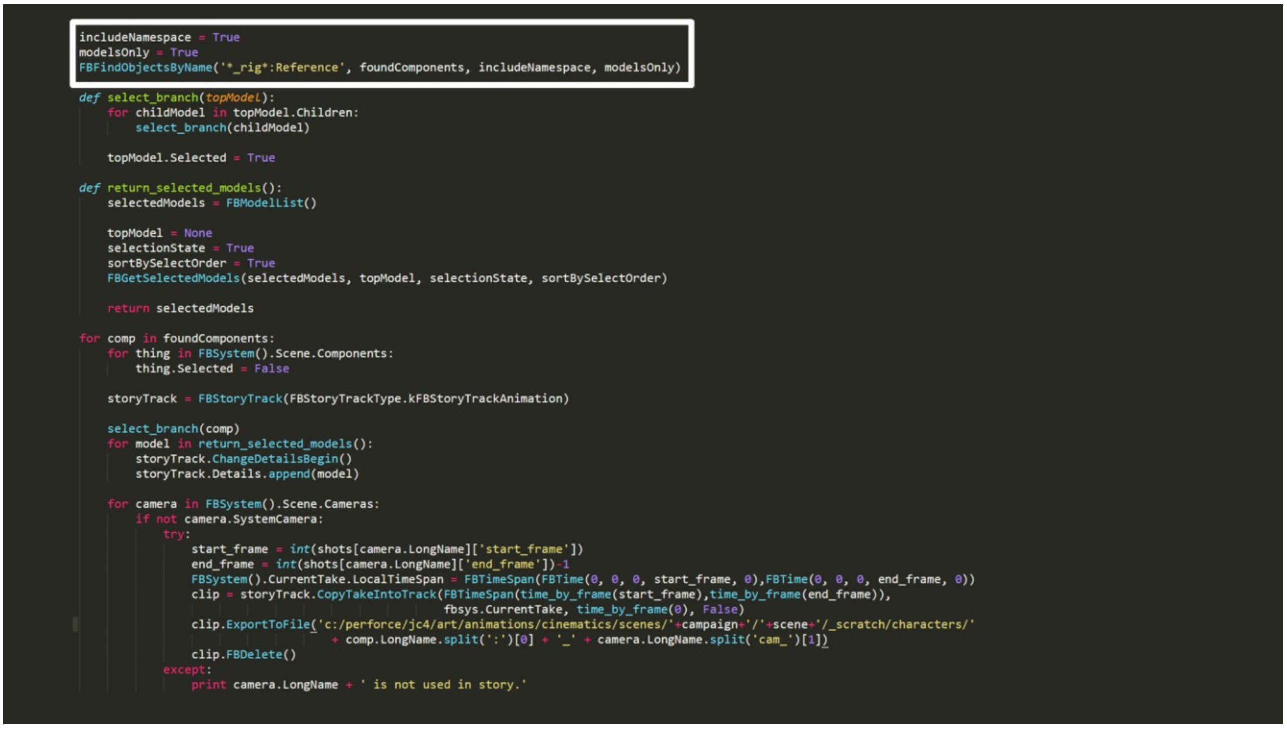
```
from pyfbsdk import *

campaign = 'sargento'
scene = 'intro_start'

foundComponents = FBComponentList()
fbsys = FBSystem()
time_by_frame = lambda x: FBTime(0, 0, 0, x)

shots = {}
allStoryTracks = FBSystem().Scene.Constraints;
for storyTrack in allStoryTracks:
    if storyTrack.LongName == "Shot Track 1":
        for clip in storyTrack.Clips:
            if len(clip.Parents):
                for lParent in clip.Parents:
                    try:
                        if not lParent.Name == 'Scene':
                            shots[clip.ShotCamera.Name] = {'start_frame':int(clip.MarkIn.GetTimeString().split('*')[0]),
                                                            'end_frame':int(clip.MarkOut.GetTimeString().split('*')[0])}
                    except:
                        pass
chars = FBSystem().Scene.Characters

plotOptions = FBPlotOptions()
timeMode = FBTimeMode.kFBTimeMode30Frames
plotOptions.PlotPeriod = FBTime(0,0,0,1,0,timeMode)
plotDest = FBCharacterPlotWhere.kFBCharacterPlotOnSkeleton

for char in chars:
    char.PlotAnimation(plotDest, plotOptions)
```

We first create a component list for everything in the scene and then gather all the start and end frame data in a list for each shot.



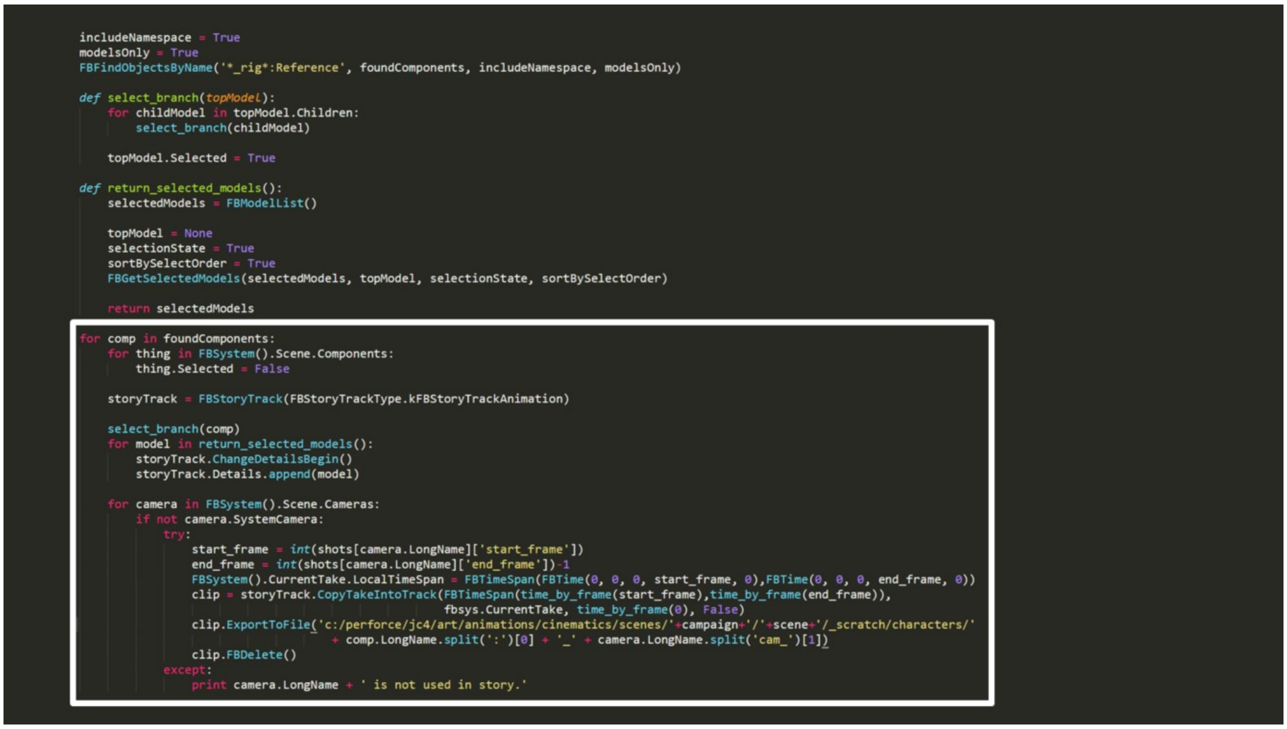
```
from pyfbsdk import *

campaign = 'sargento'
scene = 'intro_start'

foundComponents = FBComponentList()
fbsys = FBSystem()
time_by_frame = lambda x: FBTime(0, 0, 0, x)

shots = {}
allStoryTracks = FBSystem().Scene.Constraints;
for storyTrack in allStoryTracks:
    if storyTrack.LongName == "Shot Track 1":
        for clip in storyTrack.Clips:
            if len(clip.Parents):
                for lParent in clip.Parents:
                    try:
                        if not lParent.Name == 'Scene':
                            shots[clip.ShotCamera.Name] = {'start_frame':int(clip.MarkIn.GetTimeString().split('*')[0]),
                                                           'end_frame':int(clip.MarkOut.GetTimeString().split('*')[0])}
                    except:
                        pass
chars = FBSystem().Scene.Characters

plotOptions = FBPlotOptions()
timeMode = FBTimeMode.kFBTimeMode30Frames
plotOptions.PlotPeriod = FBTime(0,0,0,1,0,timeMode)
plotDest = FBCharacterPlotWhere.kFBCharacterPlotOnSkeleton

for char in chars:
    char.PlotAnimation(plotDest, plotOptions)
```

Then we plot all animation to the skeletons.



```
includeNamespace = True
modelsOnly = True
FBFindObjectsByName('*_rig*:Reference', foundComponents, includeNamespace, modelsOnly)

def select_branch(topModel):
    for childModel in topModel.Children:
        select_branch(childModel)

    topModel.Selected = True

def return_selected_models():
    selectedModels = FBModelList()

    topModel = None
    selectionState = True
    sortBySelectOrder = True
    FBGetSelectedModels(selectedModels, topModel, selectionState, sortBySelectOrder)

    return selectedModels

for comp in foundComponents:
    for thing in FBSystem().Scene.Components:
        thing.Selected = False

    storyTrack = FBStoryTrack(FBStoryTrackType.kFBStoryTrackAnimation)

    select_branch(comp)
    for model in return_selected_models():
        storyTrack.ChangeDetailsBegin()
        storyTrack.Details.append(model)

    for camera in FBSystem().Scene.Cameras:
        if not camera.SystemCamera:
            try:
                start_frame = int(shots[camera.LongName]['start_frame'])
                end_frame = int(shots[camera.LongName]['end_frame'])-1
                FBSystem().CurrentTake.LocalTimeSpan = FBTimeSpan(FBTime(0, 0, 0, start_frame, 0),FBTime(0, 0, 0, end_frame, 0))
                clip = storyTrack.CopyTakeIntoTrack(FBTimeSpan(time_by_frame(start_frame),time_by_frame(end_frame)),
                                                    fbsys.CurrentTake, time_by_frame(0), False)
                clip.ExportToFile('c:/perforce/jc4/art/animations/cinematics/scenes/'+campaign+'/'+scene+'/_scratch/characters/'
                                  + comp.LongName.split(':')[0] + '_' + camera.LongName.split('cam_')[1])
                clip.FBDelete()
            except:
                print camera.LongName + ' is not used in story.'
```

Once we've plotted, we grab the root joints of all skeletons using our component list to search from.

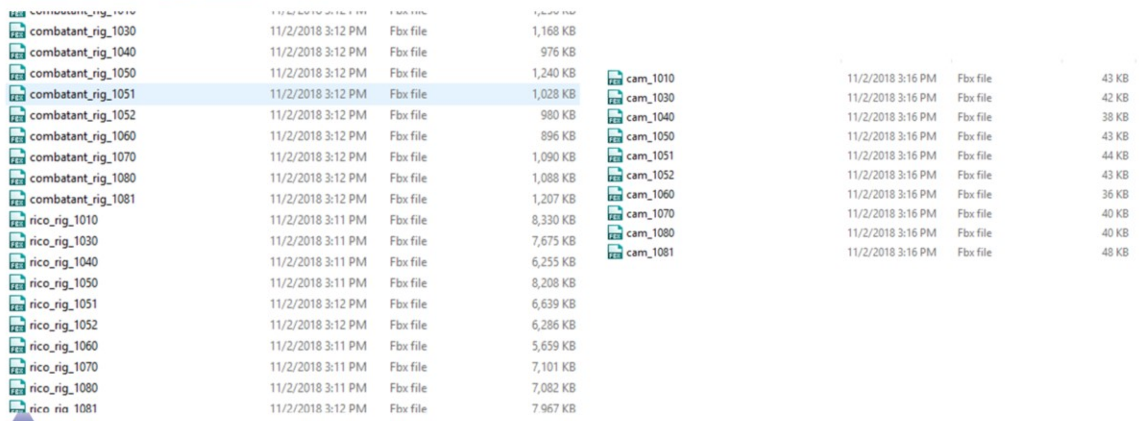


```python
includeNamespace = True
modelsOnly = True
FBFindObjectsByName('*_rig*:Reference', foundComponents, includeNamespace, modelsOnly)

def select_branch(topModel):
    for childModel in topModel.Children:
        select_branch(childModel)

    topModel.Selected = True

def return_selected_models():
    selectedModels = FBModelList()

    topModel = None
    selectionState = True
    sortBySelectOrder = True
    FBGetSelectedModels(selectedModels, topModel, selectionState, sortBySelectOrder)

    return selectedModels
for comp in foundComponents:
    for thing in FBSystem().Scene.Components:
        thing.Selected = False

    storyTrack = FBStoryTrack(FBStoryTrackType.kFBStoryTrackAnimation)

    select_branch(comp)
    for model in return_selected_models():
        storyTrack.ChangeDetailsBegin()
        storyTrack.Details.append(model)

    for camera in FBSystem().Scene.Cameras:
        if not camera.SystemCamera:
            try:
                start_frame = int(shots[camera.LongName]['start_frame'])
                end_frame = int(shots[camera.LongName]['end_frame'])-1
                FBSystem().CurrentTake.LocalTimeSpan = FBTimeSpan(FBTime(0, 0, 0, start_frame, 0),FBTime(0, 0, 0, end_frame, 0))
                clip = storyTrack.CopyTakeIntoTrack(FBTimeSpan(time_by_frame(start_frame),time_by_frame(end_frame)),
                                                    fbsys.CurrentTake, time_by_frame(0), False)
                clip.ExportToFile('c:/perforce/jc4/art/animations/cinematics/scenes/'+campaign+'/'+scene+'/_scratch/characters/'
                                  + comp.LongName.split(':')[0] + '_' + camera.LongName.split('cam_')[1])
                clip.FBDelete()
            except:
                print camera.LongName + ' is not used in story.'
```

And we iterate though all that making sure to grab the entire hierarchy of each individual skeleton as we create separate clips per character and export out a separate file per character per shot.

JUST CAUSE 4

Cinematic Content

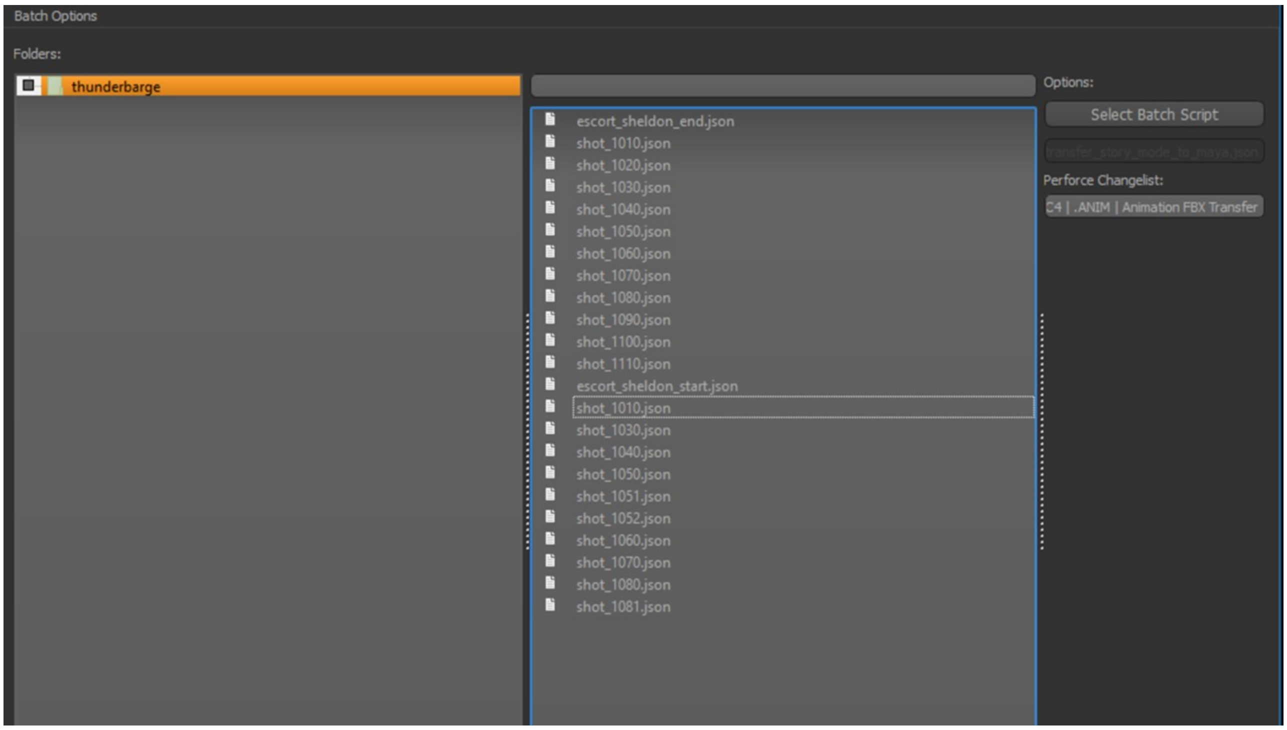


```
combatant_rig_1010     11/2/2018 3:12 PM   Fbx file   [illegible]
combatant_rig_1030     11/2/2018 3:12 PM   Fbx file   1,168 KB
combatant_rig_1040     11/2/2018 3:12 PM   Fbx file     976 KB
combatant_rig_1050     11/2/2018 3:12 PM   Fbx file   1,240 KB
combatant_rig_1051     11/2/2018 3:12 PM   Fbx file   1,028 KB
combatant_rig_1052     11/2/2018 3:12 PM   Fbx file     980 KB
combatant_rig_1060     11/2/2018 3:12 PM   Fbx file     896 KB
combatant_rig_1070     11/2/2018 3:12 PM   Fbx file   1,090 KB
combatant_rig_1080     11/2/2018 3:12 PM   Fbx file   1,088 KB
combatant_rig_1081     11/2/2018 3:12 PM   Fbx file   1,207 KB
rico_rig_1010          11/2/2018 3:11 PM   Fbx file   8,330 KB
rico_rig_1030          11/2/2018 3:11 PM   Fbx file   7,675 KB
rico_rig_1040          11/2/2018 3:11 PM   Fbx file   6,255 KB
rico_rig_1050          11/2/2018 3:11 PM   Fbx file   8,208 KB
rico_rig_1051          11/2/2018 3:12 PM   Fbx file   6,639 KB
rico_rig_1052          11/2/2018 3:12 PM   Fbx file   6,286 KB
rico_rig_1060          11/2/2018 3:11 PM   Fbx file   5,659 KB
rico_rig_1070          11/2/2018 3:11 PM   Fbx file   7,101 KB
rico_rig_1080          11/2/2018 3:11 PM   Fbx file   7,082 KB
rico_rig_1081          11/2/2018 3:12 PM   Fbx file   7,967 KB
```

```
cam_1010     11/2/2018 3:16 PM   Fbx file   43 KB
cam_1030     11/2/2018 3:16 PM   Fbx file   42 KB
cam_1040     11/2/2018 3:16 PM   Fbx file   38 KB
cam_1050     11/2/2018 3:16 PM   Fbx file   43 KB
cam_1051     11/2/2018 3:16 PM   Fbx file   44 KB
cam_1052     11/2/2018 3:16 PM   Fbx file   43 KB
cam_1060     11/2/2018 3:16 PM   Fbx file   36 KB
cam_1070     11/2/2018 3:16 PM   Fbx file   40 KB
cam_1080     11/2/2018 3:16 PM   Fbx file   40 KB
cam_1081     11/2/2018 3:16 PM   Fbx file   48 KB
```

Our results look like this and include file sizes that range from mere KBs for cameras and anywhere from about 0.5 mb and 10mb per character depending on complexity of skeleton and length of shot (Rico has the largest files and background characters have much less). MUCH more manageable now that we’ve broken this down!

These files truly contain just skeletons with baked animation data. They look exactly the same as the skeleton only animation files that get created from that batch export takes tool I showed. Shot by shot we can now open up a fresh Maya file, reference in only the rigs for the assets that actually exist in each specific shot, and it only takes a matter of seconds to reference these small files in which then we transfer the animation onto the Maya rigs from, which then obviously takes a bit more time than just mere seconds to do.



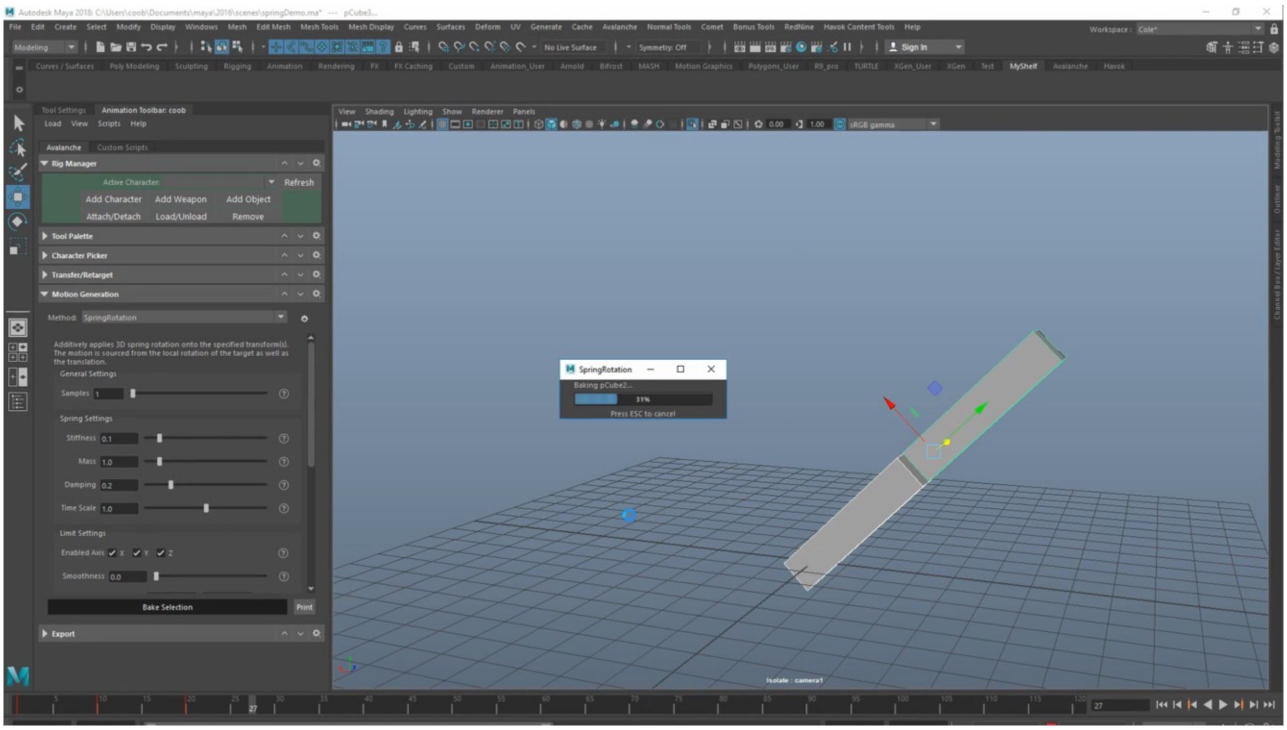
transfer_story_mode_to_maya.json

Perforce Changelist:

C4 | .ANIM | Animation FBX Transfer

```
escort_sheldon_end.json
shot_1010.json
shot_1020.json
shot_1030.json
shot_1040.json
shot_1050.json
shot_1060.json
shot_1070.json
shot_1080.json
shot_1090.json
shot_1100.json
shot_1110.json
escort_sheldon_start.json
shot_1010.json
shot_1030.json
shot_1040.json
shot_1050.json
shot_1051.json
shot_1052.json
shot_1060.json
shot_1070.json
shot_1080.json
shot_1081.json
```

And it's all streamed through the batch tool! The transfer script for taking these exported pieces from MoBu does all the referencing, baking, and file cleanup in a snap! We can process multiple shots, scenes, whatever much more quickly than by using default built in transfer setups.



Method: SpringRotation

Additively applies 3D spring rotation onto the specified transform(s). The motion is sourced from the local rotation of the target as well as the translation.

General Settings

Samples 1

Spring Settings

Stiffness 0.1
Mass 1.0
Damping 0.2
Time Scale 1.0

Limit Settings

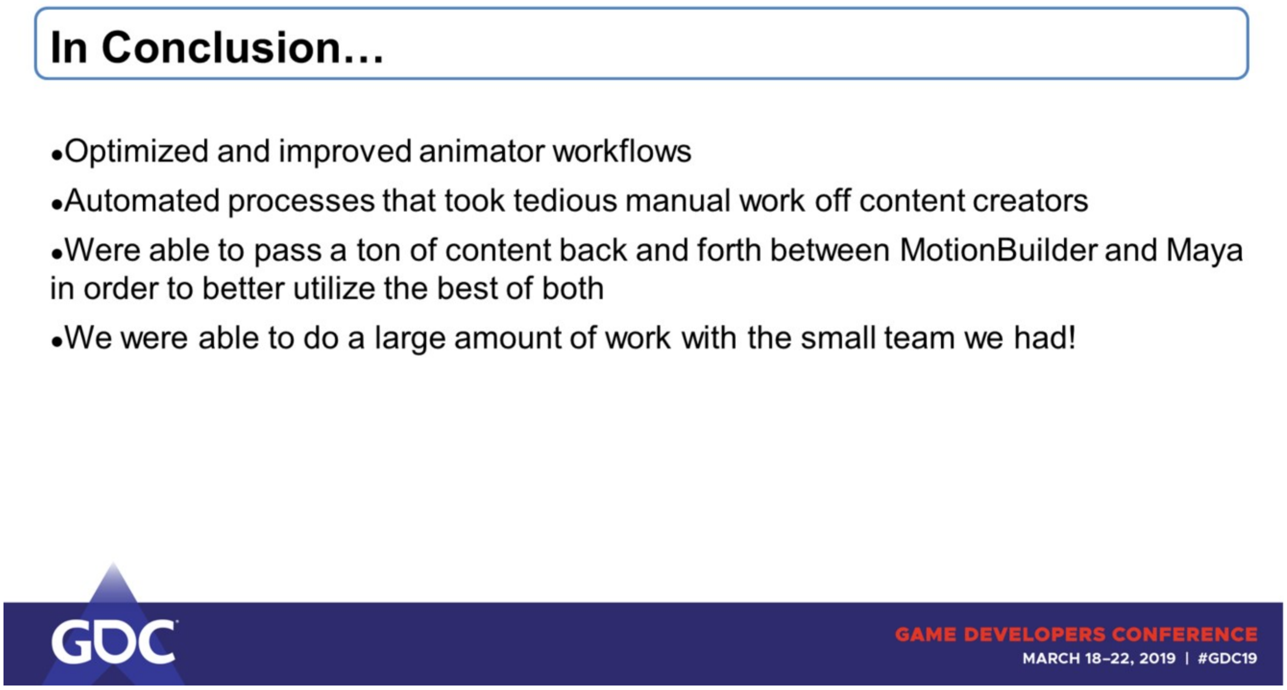
Enabled Axis ✓ X ✓ Y ✓ Z
Smoothness 0.0

Baking pCube2...
31%
Press ESC to cancel

I mentioned earlier that Maya was necessary for secondary animation polish and here's why.

One awesome content tool we had in Maya was the "motion generation" tool. It could use a number of different solvers from Maya dynamics such as a spring or bounce and then take in some user settings to then process that based off the root motion of the characters. We could batch through entire cutscenes in Maya and run an auto-secondary animation pass on all characters very quickly. I think our first pass only took a week to add secondary animation to ALL cutscenes.

The extra special thing about thing tool is that we could run it through a playblast script we had in conjunction with batch and we would get a directory of MP4 files of each Maya file/shot to see the secondary animation. This was really useful for going in and tweaking specific shots, whether it be for secondary motion, cameras, or characters.



In Conclusion…

- Optimized and improved animator workflows
- Automated processes that took tedious manual work off content creators
- Were able to pass a ton of content back and forth between MotionBuilder and Maya in order to better utilize the best of both
- We were able to do a large amount of work with the small team we had!

GDC

GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

To wrap this all up, what we were able to accomplish included…

-Better Workflows

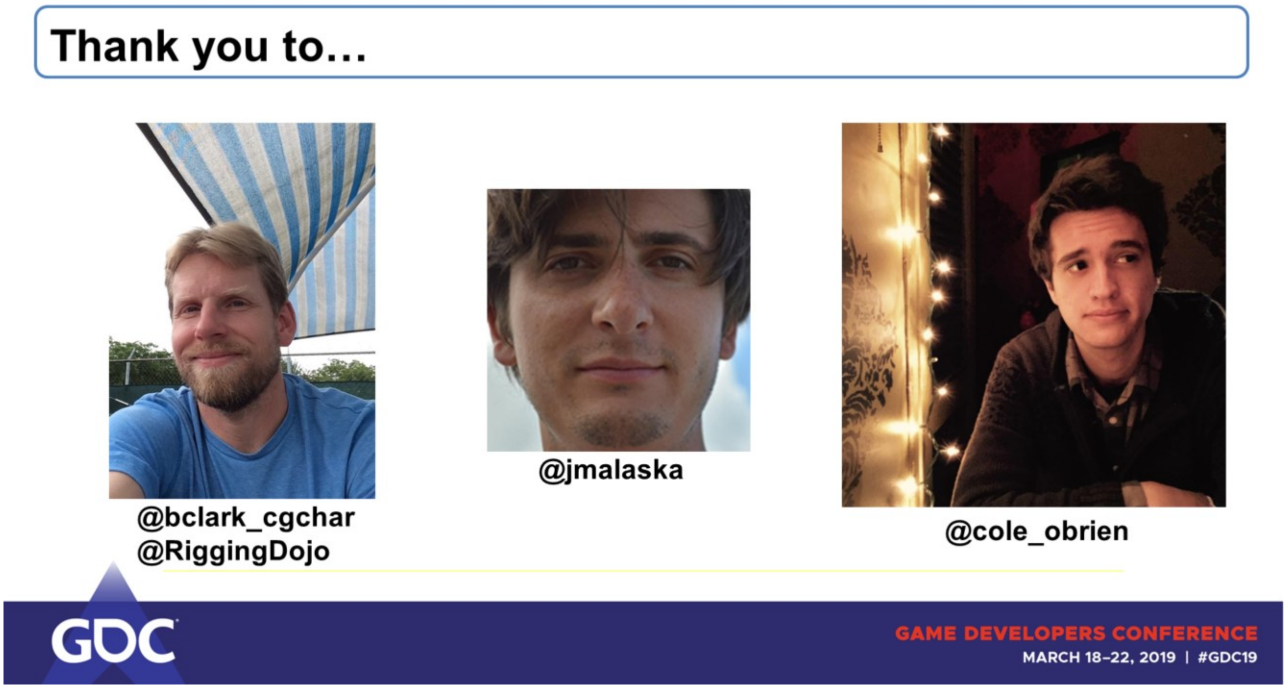
-More power and customization

-A lot more automation and less manual work

-A full 360 degree pipeline between Maya and MotionBuilder

-And all of this helped us churn out a ton of content that wouldn't have otherwise been possible unless we had a small army to help us

-And while there are other studios that have some similar tools and workflows, I hope that this talk at the very least gave a new perspective to those of you that are familiar, and for the rest I hope that you've enjoyed a bit of the behind the scenes and under the hood that I've presented to you



Thank you to…

@bclark_cgchar
@RiggingDojo

@jmalaska

@cole_obrien

Before I REALLY conclude, I want to extend a huge thank you to these guys who all helped me in some way to develop the core of a lot of the things I've presented.

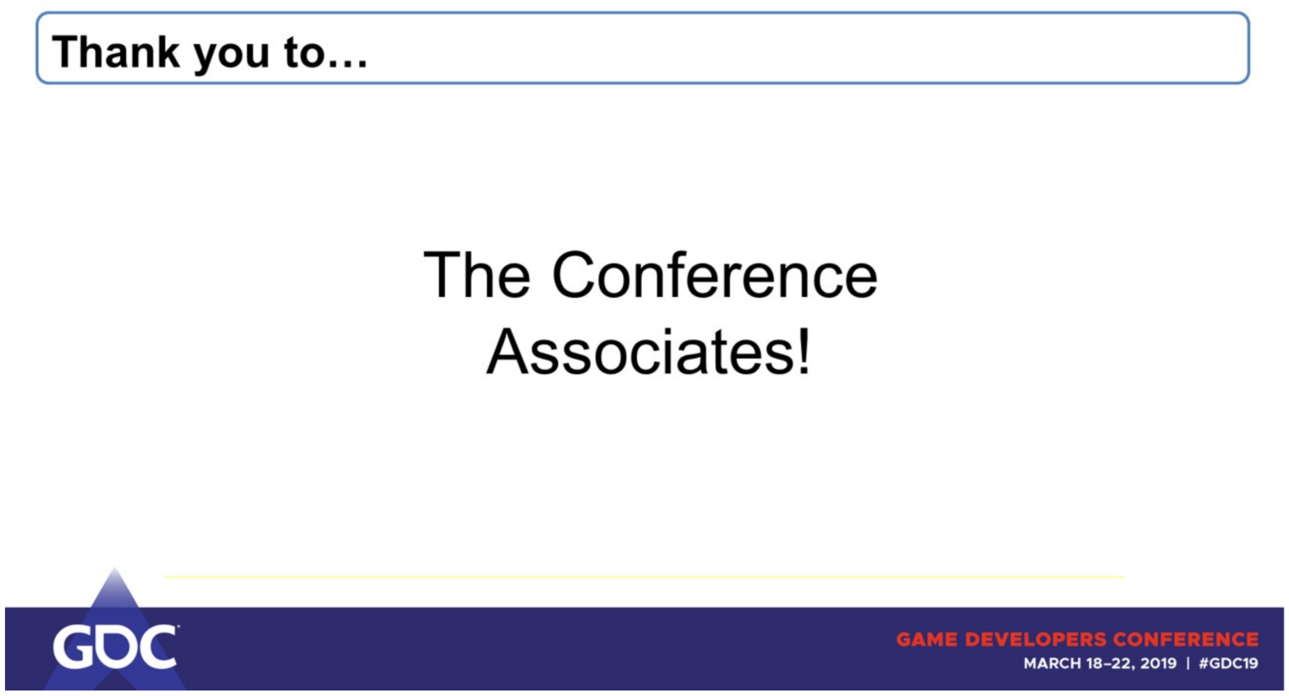
Brad Clark, co-founder of Rigging Dojo there on the left was always available to help answer any specific MoBu question I had, and he still does anytime I have anything to ask. He also made a quick demo video just for me on how the camera switcher work.

John Malaska in the middle here, who actually gave a talk this past Monday at the animation bootcamp about freelancing for animators (check it out on the vault if you didn't see), he come on board for a few months when I was cold, alone, and in desperate need of TA help. He was integral in a lot of the MoBu takes script work.

Finally Cole O'Brien on the right was responsible for many Maya optimizations including being a big part of the custom script stuff as well as the motion generation tool you saw.
You can find them all on twitter with these handles.

Also, they aren't listed or shown here but I love my animation team, they're awesome and despite all their demands, they really make me a better TA.

Also thank you to..



Thank you to…

The Conference Associates!

The CAs! They work so hard all week, don’t get sleep, always smile, and are genuinely the nicest people in the world…GDC wouldn’t be GDC without your volunteering so thank you so much!

Thank you to…

YOU!
Twitter: @TechAnimator
Email: brian@techanimator.com

And thank YOU so much for coming out to my talk today! (If time permits have Q&A) I'll stick around outside in the designated post-talk area if you have any more questions.