

FARCRY[®]

PRIMAL

CHARACTER PIPELINE & CUSTOMIZATION SYSTEM

Julien Lallevé @jlallevé
Character Tech Programmer

Kieran O'Sullivan @kizzafreak
Character Technical Director

DEFINITIONS

DCC : Digital Content Creation (software)

- 3DS Max, Maya, Blender, MotionBuilder...

DUNIA

- The game engine of the Far Cry brand

FARCRY
PRIMAL

DEFINITIONS

Wolfskin

- Name of our new system to assemble and customize characters

DNA

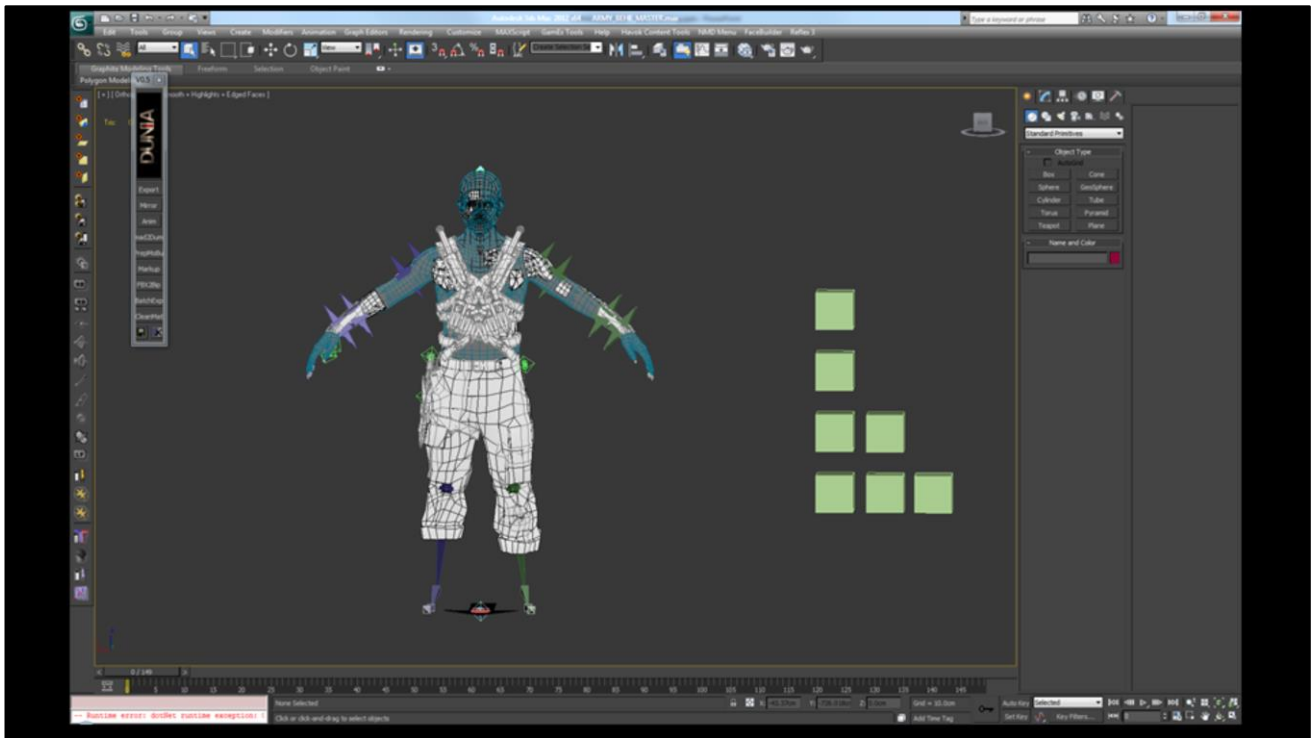
- Piece of Wolfskin data that represents a full character

FARCRY
PRIMAL



[Kieran]

To begin we will first talk about the pipeline and tools from Far Cry 3 & 4, and the issues we wanted to solve for this new game.



[Kieran]

Artists were working in large 3DS Max 2012 scenes containing multiple assets. As you can see the shaders aren't doing anything, on Far Cry 3 most of our shaders didn't work or wouldn't be kept up to date with the engine.

In this example you can see we have multiple upper body parts, and maybe one or two pants.

But you can also see we don't have a complete character, we have no legs or shoes in this file, so we probably don't have all the parts we need to work.

OLD DUNIA MODELING PIPELINE



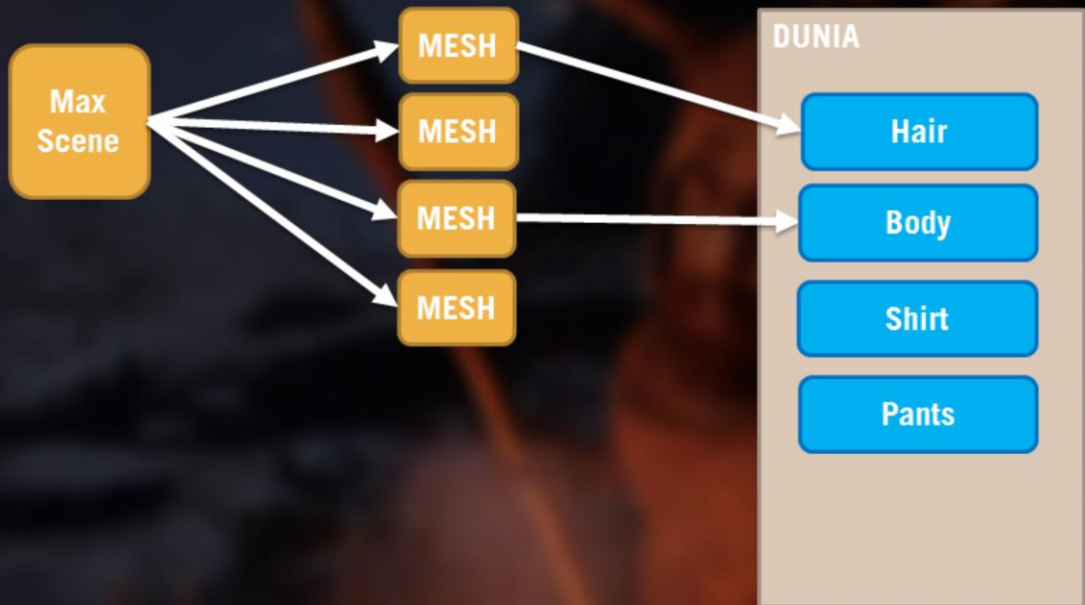
FARCRY
PRIMAL

[Kieran]

A single Max file will contain multiple meshes that are exported to the engine.

This is a conservative example, often there will be several, or dozens, of meshes saved in one Max file.

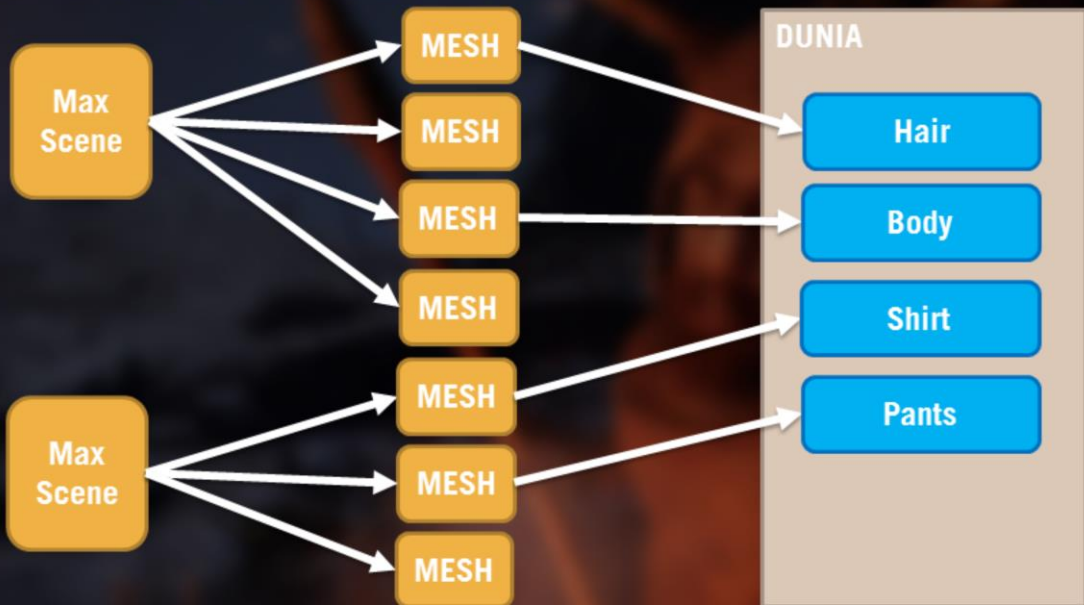
OLD DUNIA MODELING PIPELINE



[Kieran]

Some of these meshes will be assigned to the character in the engine.
In this case it's the hair and body.

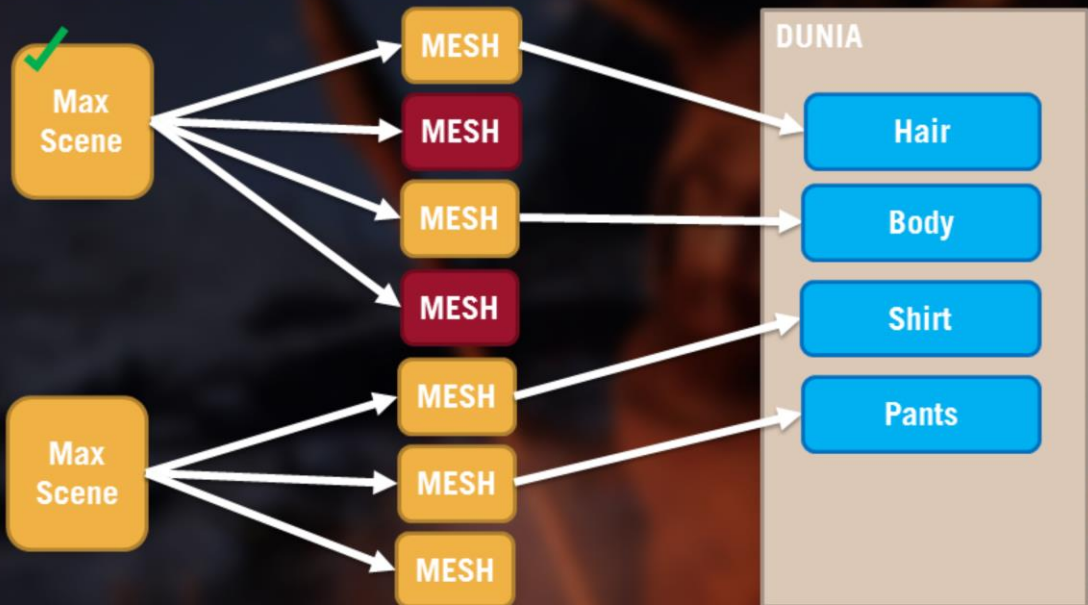
OLD DUNIA MODELING PIPELINE



[Kieran]

But we use the shirt and pants from other files, which were exported from a different Max source file.

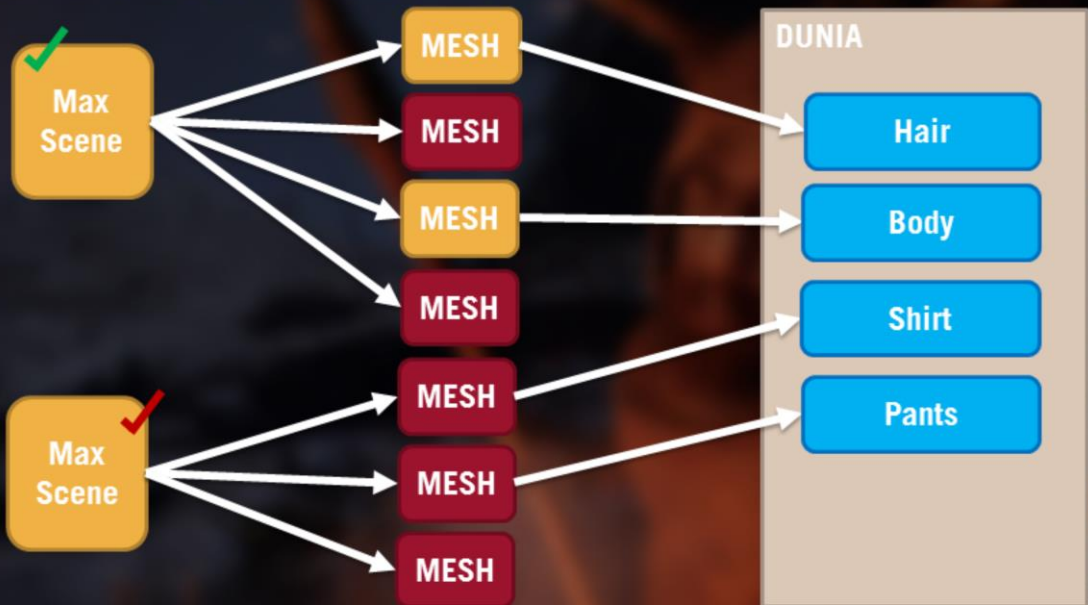
OLD DUNIA MODELING PIPELINE



[Kieran]

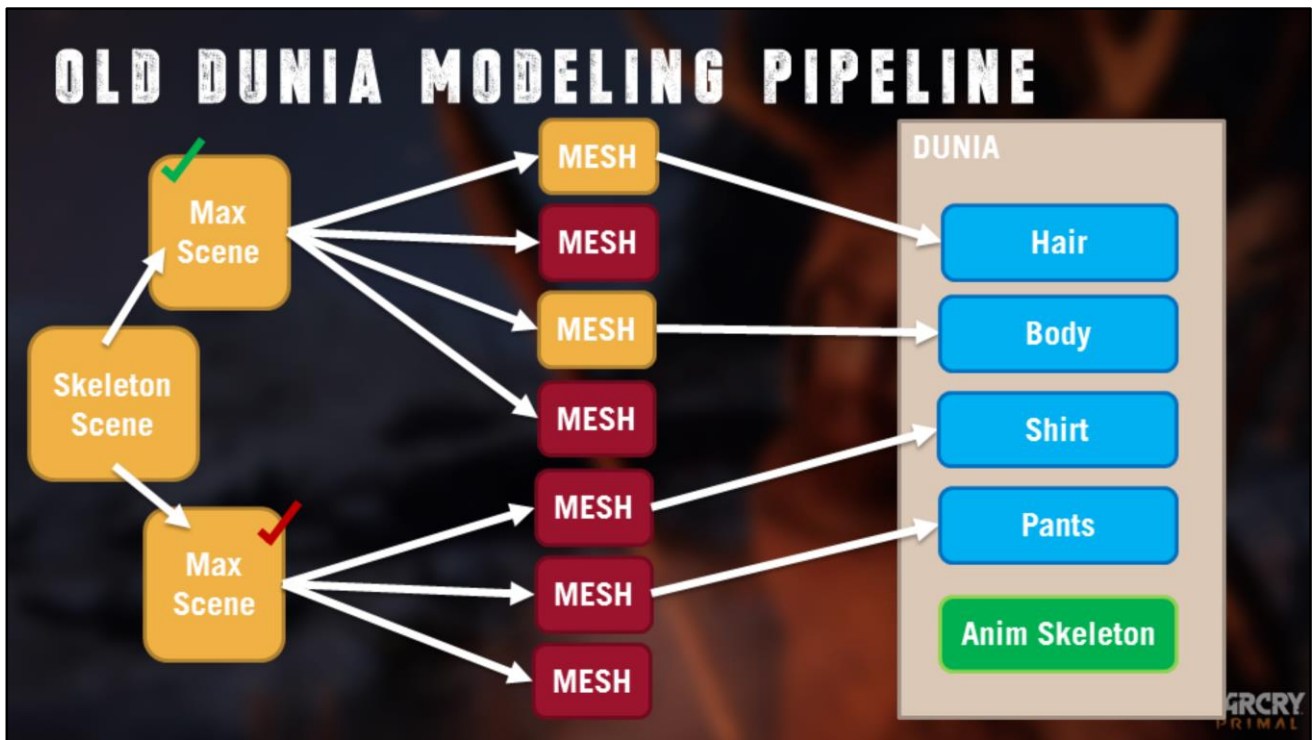
If I want to work on these meshes I will check out the source Max file from Perforce. This means no one can work on meshes from that scene, even the meshes that I don't actually need to edit.

OLD DUNIA MODELING PIPELINE



[Kieran]

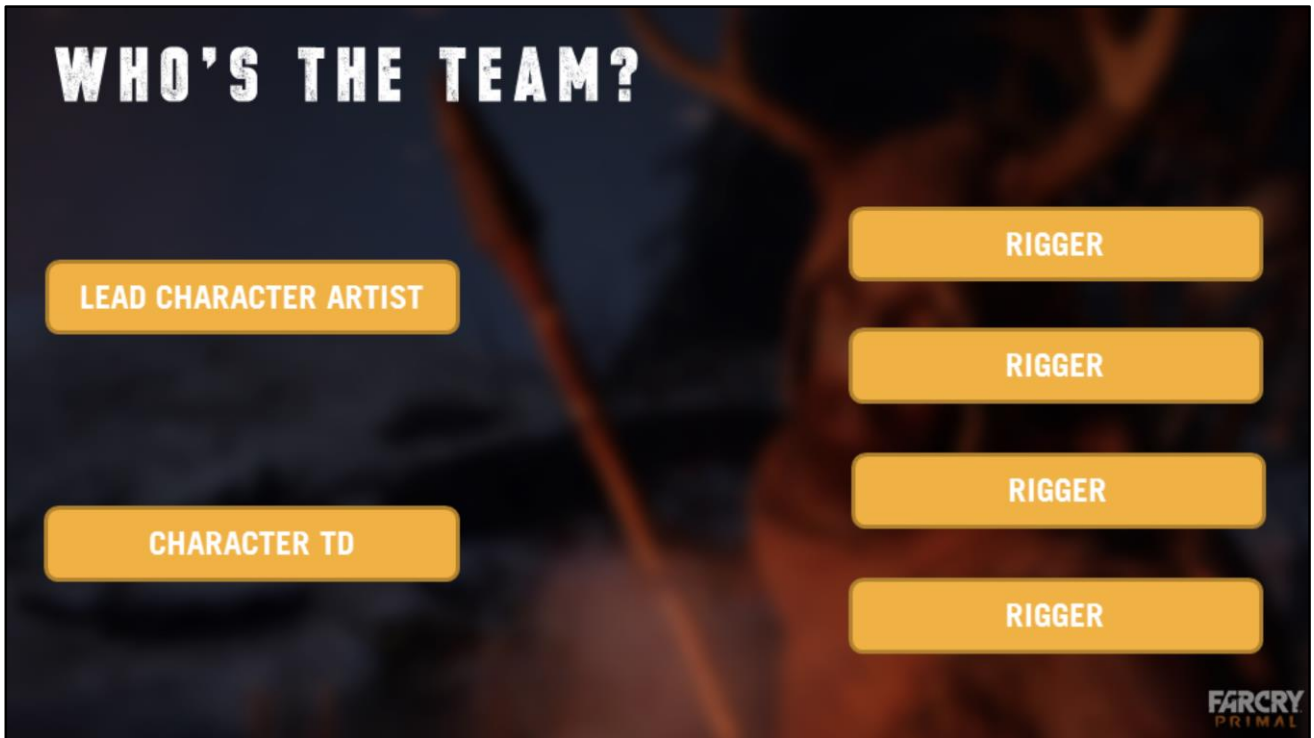
Even worse, if someone else has checked out a Max file I can't work on **any** meshes that came from this source file.



[Kieran]

To make things more complicated, we need to keep the skeleton in sync.

So we have an extra Max file that contains the skeleton and we merge it into our working files. If we ever need to change the skeleton we need to update all the Max files... so we don't do that.



[Kieran]

Let's pause for a moment and look at who's in the character team, working directly with the pipeline.

(Leaving out modelers who sent their meshes to riggers for integration)

WHO'S THE TEAM?

FAVORITE DCC

LEAD CHARACTER ARTIST



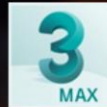
CHARACTER TD



RIGGER



RIGGER



RIGGER



RIGGER

FARCRY
PRIMAL

[Kieran]

An artists's favorite DCC is where they feel more comfortable working. It's the software that's gonna let them get the most out of their skills.

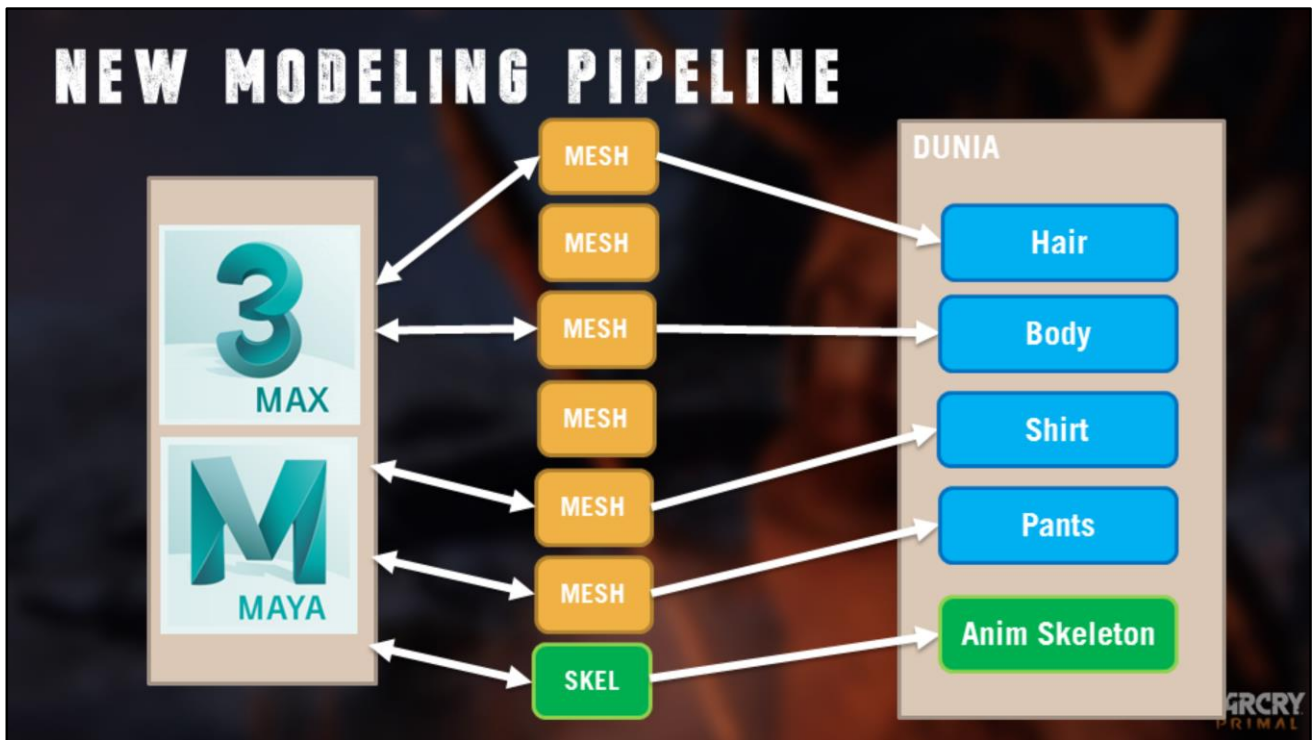
Getting as close as possible from 100% efficiency for your artists makes all the difference.

NEW MODELING PIPELINE



[Kieran]

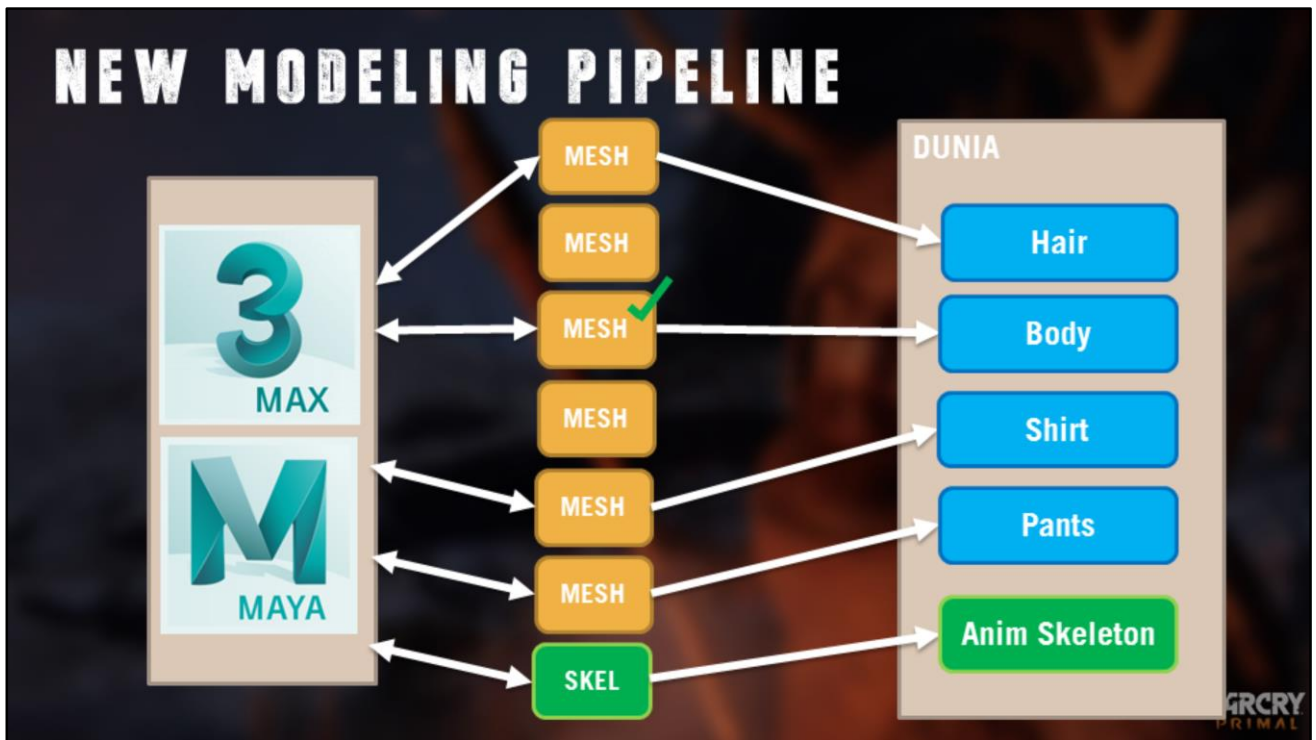
So, what would be our ideal pipeline? Well first, I'd like to get rid of all those source files!



[Kieran]

If I want to edit these meshes, I want to open **all** of them at the same time, ideally in either 3DS Max, or in Maya. As we've seen, Half our character team were Maya users on Primal.

And I want to always get the latest version of the skeleton automatically as well.



[Kieran]

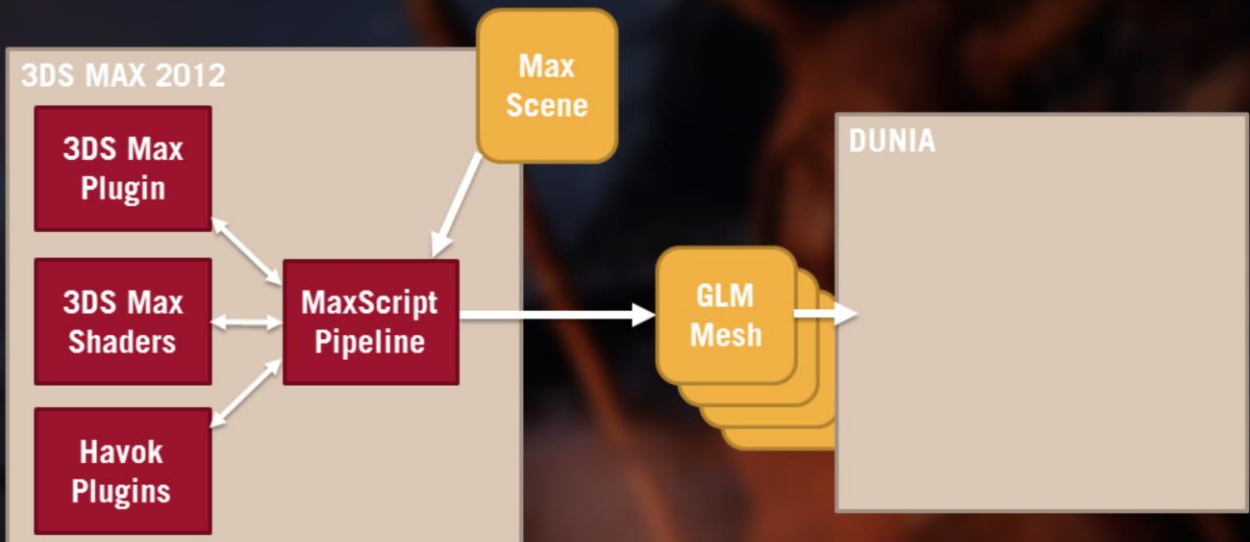
When I export a file back to the engine, the mesh gets checked out of Perforce.

But none of the other files are checked out, unless I need to edit them.

So I'm not going to block anyone from editing the other files, and no one will block me.

This makes it really easy to work in parallel, even on different parts of the same character!

OLD DUNIA MODELING PIPELINE

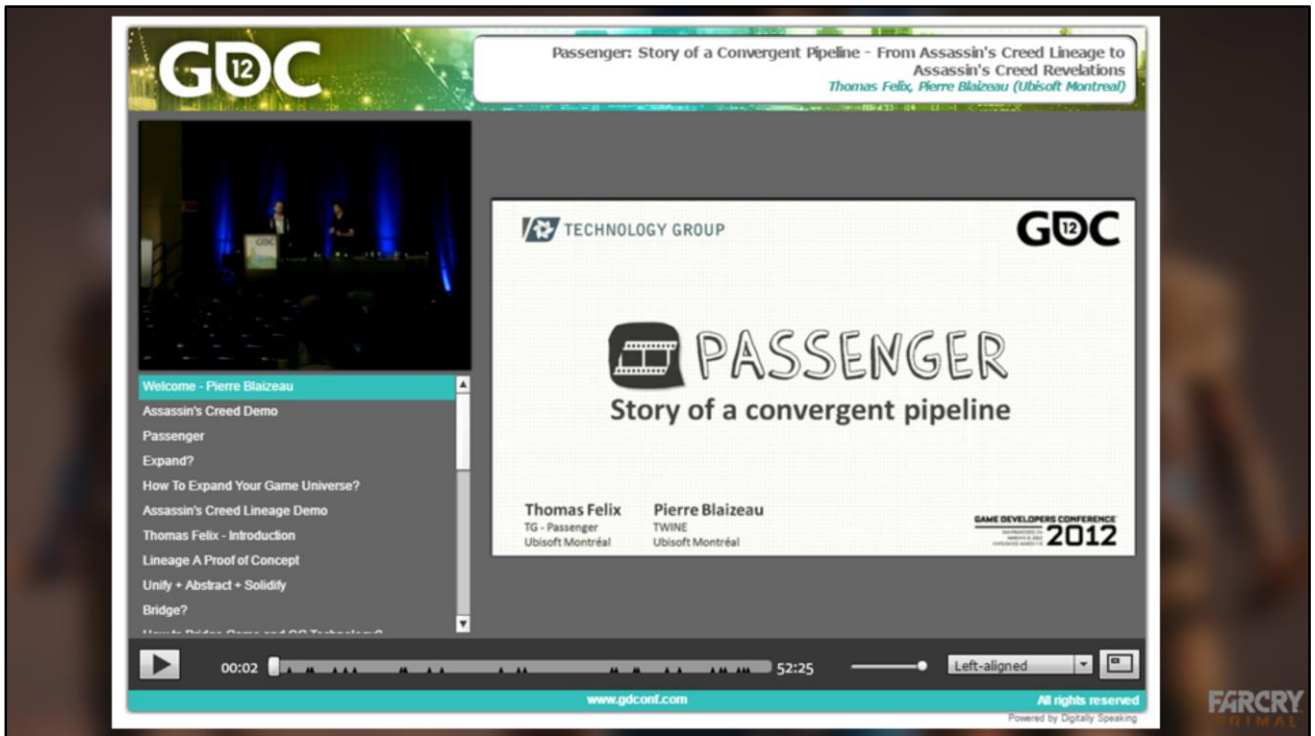


[Kieran]

Now we know how we want to work, let's take a look at the tools.

[Julien]

Going back to the old pipeline you can see we had a source file in 3DS Max 2012, and used a Maxscript pipeline that depends on Max plugins, Max Shaders, and Havok plugins. Through all that we export a mesh, or multiple meshes, which are then loaded into the engine.



[Julien]

Few years ago, the Ubisoft Technology group developed the Passenger framework : a tool to extract game assets into DCCs for marketing needs. This led to a suite of tools for serializing/deserializing assets such as meshes or skeletons from the game to Max, Maya or MotionBuilder. The format for these assets is called Gamex.

Basically, we now had an extendable asset format with APIs in Max, Maya, Mobu and our game engine. It wasn't exactly a modelling pipeline yet, but the question was, could it be ? By making the source asset in engine be gamex assets, we would benefit from back and forth workflow between the engine and any dcc.

If you want to know more about the origins of Passenger and Gamex, you can check out that GDC12 presentation freely available on GDC vault.



Using the passenger asset extraction framework as the basis for a modeling pipeline.

GAMEX MODELING PIPELINE



[Julien]

Instead of a source Max file and engine Mesh files, we just have 1 Gamex file per asset.

This file can be opened in 3DS Max, Maya, as well as the game engine.

The Technology Group creates and maintains import/export plugins for Max, Maya, and other applications. As well there are Python, C#, C++ APIs for dealing with Gamex data.

GAMEX MODELING PIPELINE

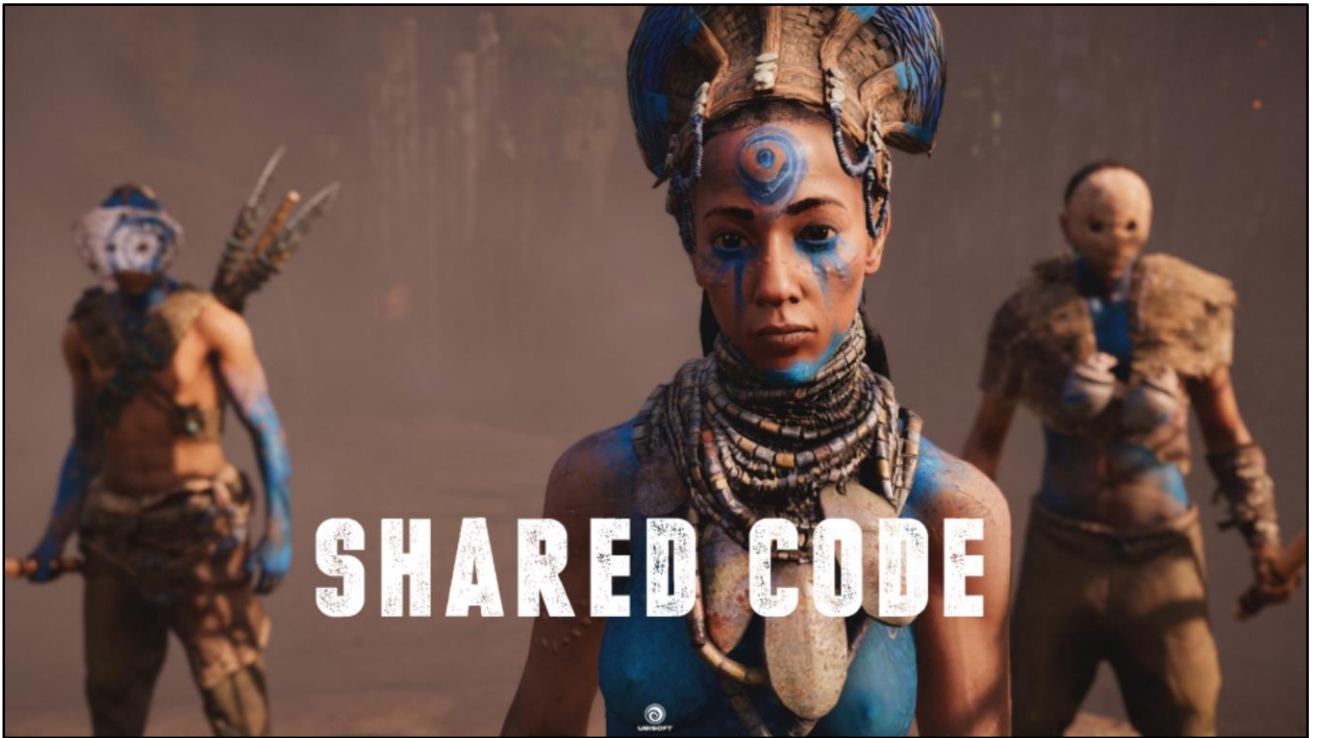


[Julien]

The Passenger team provided a plugin to serialize/deserialize assets in Max and Maya. On top of that, we built an import/export pipeline customized for the needs of our game engine.

With the vision to support Max and Maya, we decided to write a single shared pipeline, and made use of common technology such as Python, PySide for UI, or ShaderFX for materials.

Notice the back and forth between the asset file, and the absence of any scene that would lock several assets together.

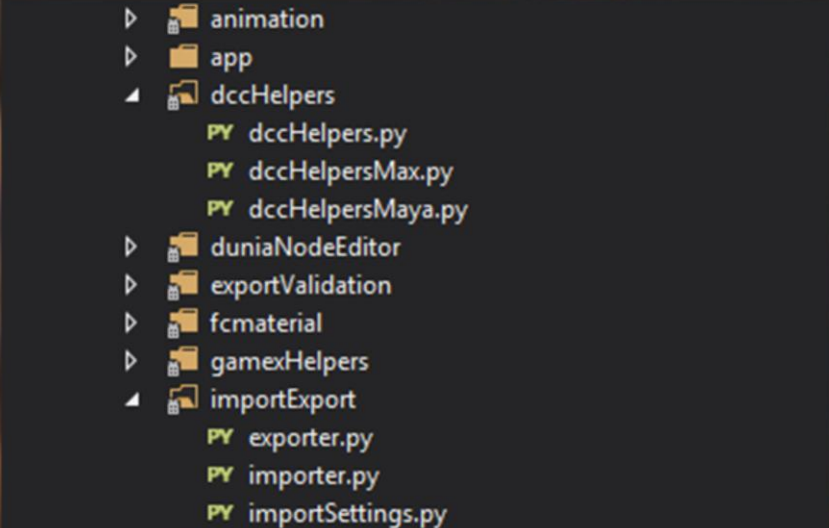


[Julien]

Supporting two pipelines for Max AND Maya is unrealistic, we had to share it.

However, those 2 software have very different APIs, they don't even originate from the same company! So how do we share code ?

SINGLE MAX/MAYA PIPELINE



```
└─ animation
└─ app
└─ dccHelpers
    ├── PY dccHelpers.py
    ├── PY dccHelpersMax.py
    └── PY dccHelpersMaya.py
└─ duniaNodeEditor
└─ exportValidation
└─ fcmaterial
└─ gamexHelpers
└─ importExport
    ├── PY exporter.py
    ├── PY importer.py
    └── PY importSettings.py
```

FARCRY
BLOOD DRAGON

[Julien]

If you look at the file structure of the pipeline, you'll find that files are split in two groups.

On one side you have DCC agnostic code, which covers all high level logic in the pipeline. How to export/import meshes, how to bind materials from the game engine etc.

On the other side, you have DCC specific code for Max and Maya which deals with all low level functionality.

DCC AGNOSTIC HIGH LEVEL

```
def exportGeometry( mode, validate, treatAsNew = False ) :  
  
    selection = dccHelpers.getSelectedNodes( mode )  
  
    try:  
        # Filter selection to interesting nodes  
        nodes = getAssetNodesFromSelection(selection, True)  
        if not nodes or len(nodes) == 0 :  
            dccHelpers.showErrorWindow('Export Error', "No valid objects Selected")  
            return False  
  
        sceneLib = sceneLibrary.SceneLibrary()  
        sceneLib.addAllSceneObjects()  
  
        for node in nodes:  
  
            print "Exporting {0} ...".format(node)
```

[Julien]

Let's look at an example.

This is the beginning of the logic to export selected geometries to the game. The code doesn't know in what DCC it's running as there's no point duplicating that logic between Max and Maya. Here we start by getting the selection from the user to know what we need to export, we filter that selection a bit, and then go on to exporting each asset individually.

Let's look at the dccHelpers module we call here to get the user selection, a low level task.

SPECIFIC LOW LEVEL

```
if DCCPlatform.name == "Maya":
    import dccHelpersMaya as localDccHelpers
elif DCCPlatform.name == "3dsmax":
    import dccHelpersMax as localDccHelpers

def getSelectedNodes():
    return localDccHelpers.getSelectedNodes()

def select(dccNodes):
    return localDccHelpers.select(dccNodes)
```

FARCRY
PRIMAL

[Julien]

There's no common way of getting the selection in both Max and Maya, so in our helper module, we use a conditional import to branch code in either of the DCC specific version.

The DCC's name here as exposed by Passenger's plugin is our hint for which file we want to import.

SPECIFIC LOW LEVEL



```
def getSelectedNodes():  
    nodeList = []  
    for node in MaxPlus.SelectionManager.Nodes:  
        nodeList.append( gx.dcc.DCCNode( node ) )  
    return nodeList
```



```
def getSelectedNodes():  
    nodeList = []  
    selection = cmds.ls( selection=1, long=True )  
    for node in selection:  
        nodeList.append( gx.dcc.DCCNode ( node ) )  
    return nodeList
```

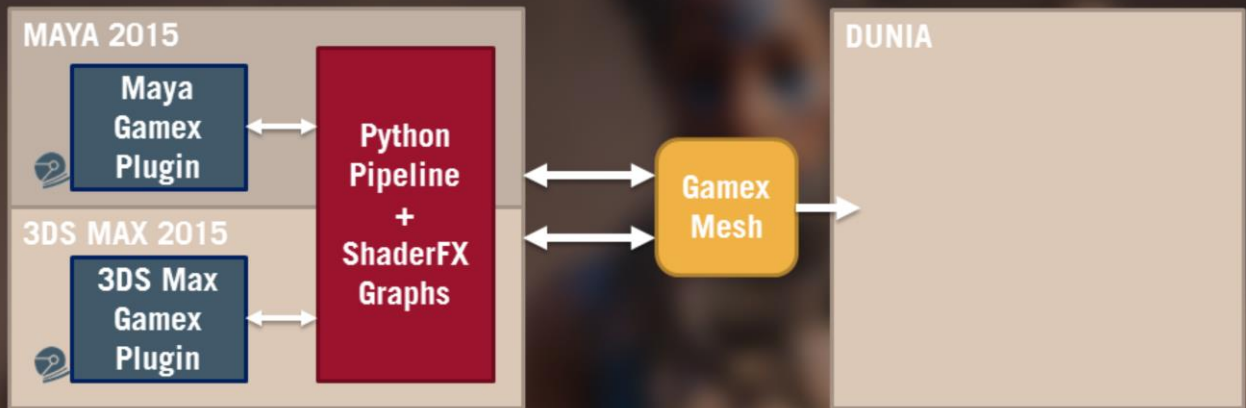
FARCRY
PRIMAL

[Julien]

We can see here that we have two versions of the get selection function, one Max specific with MaxPlus in the dccHelpersMax file, and the other one with Maya cmds in dccHelpersMaya.

Because nodes are different kind of objects in Max and Maya too, we use a wrapper class to send it back to the dcc agnostic logic layer, so that the high level doesn't need to worry on the API of the objects it's dealing with.

GAMEX MODELING PIPELINE



[Julien]

So that's how we share code. We have a very large part of the pipeline in the DCC agnostic logic, and we build a library of DCC specific low level helpers as we need them. After a while, most of the new functionalities rely on that low level layer, and can be added without a single line of DCC specific code, making the pipeline very cheap to maintain in both softwares. Enabling programmers to focus on feature logic rather than Max/Maya specificity of API.



[Julien]

- Pipeline needs UI for artists
- Sharing UI between max and maya ?

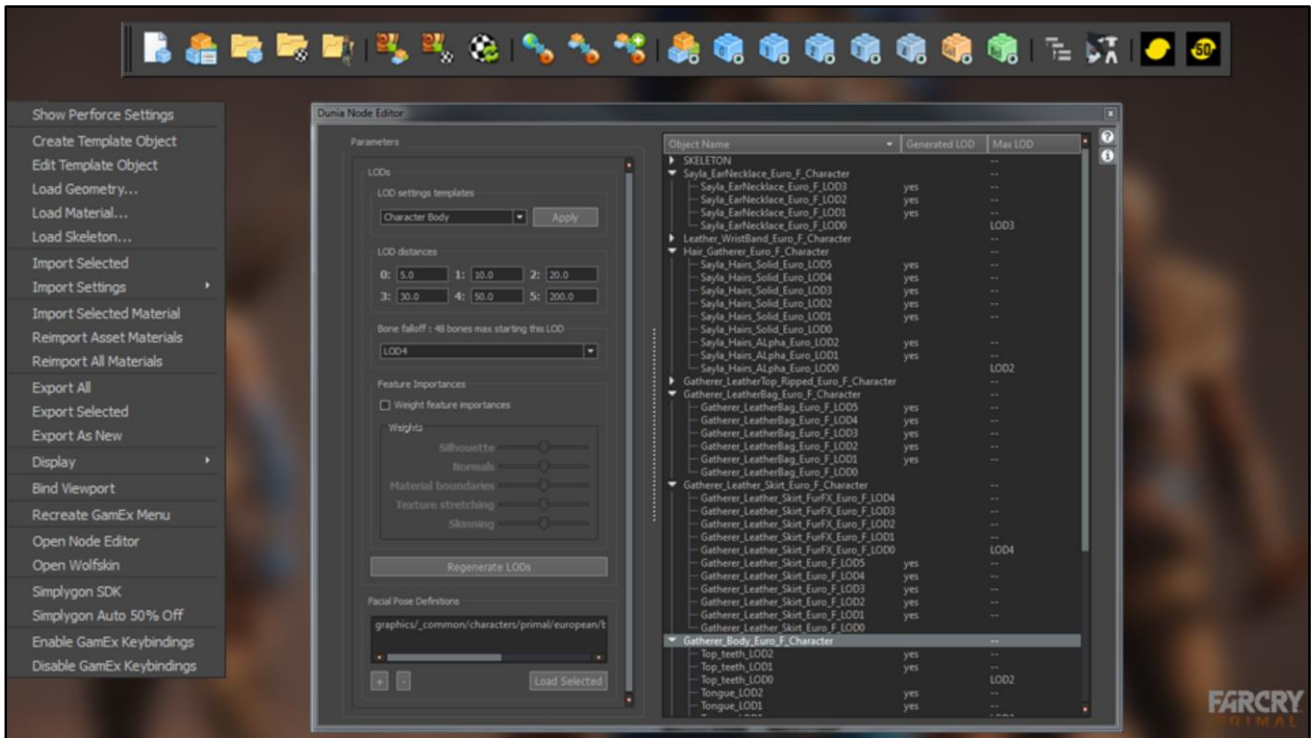
Version	DCC	Python	Pyside
2014	Max	~	✓
	Maya	✓	✓
2015	Max	✓	✓
	Maya	✓	✓
2016	Max	✓	✓
	Maya	✓	✓
2017	Max	✓+	✓
	Maya	✓	✗ Pyside2

FARCRY
PRIMAL

[Julien]

A lot of Autodesk products include PySide, the python binding for QT 4.

When we started Primal production, 2015 was the current version so having PySide in both made it an easy choice for UI needs.



[Kieran]

All of our tools are written in Python, with the UI built in QT designer, so we can have the same tools with the same behavior in both Max and Maya.

In this example you can see our menu and tool bars, which are all identical in both packages.

In the middle is our Node Editor, where we can set LOD distances per mesh, as well as LOD settings.

With the help of the Tools Group we integrated Simplygon into our pipeline for automatically generating LODs. The artist can edit the LOD settings in the node editor, and our tools will use GamEx to send and receive the LODs back from Simplygon. When LOD meshes are built by Simplygon they receive metadata so we can keep track of which LODs are generated and which are built by hand.



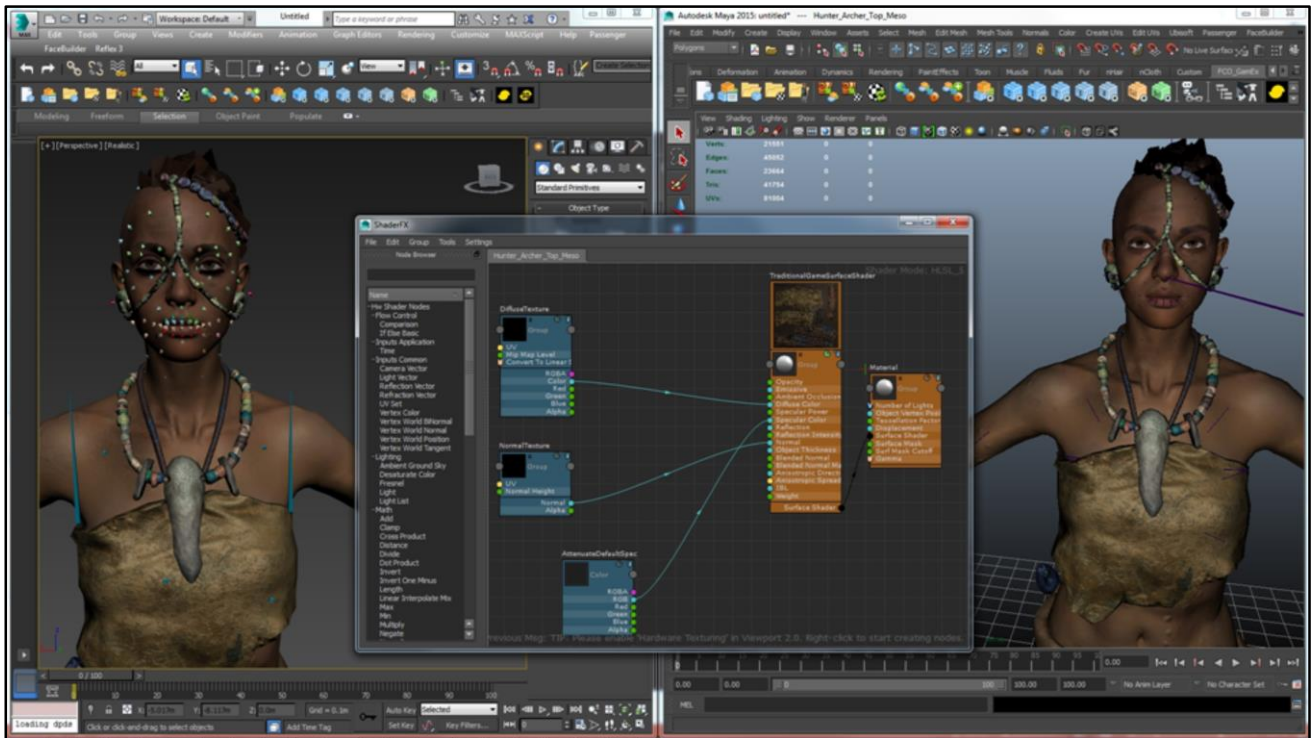
[Kieran]
- Shaders!

Version	DCC	ShaderFX	Stingray
2014	Max	×	×
	Maya	×	×
2015	Max	✓~	×
	Maya	✓	×
2016	Max	✓	✓
	Maya	✓	✓
2017	Max	✓	✓
	Maya	✓	✓

FARCRY
PRIMAL

[Kieran]

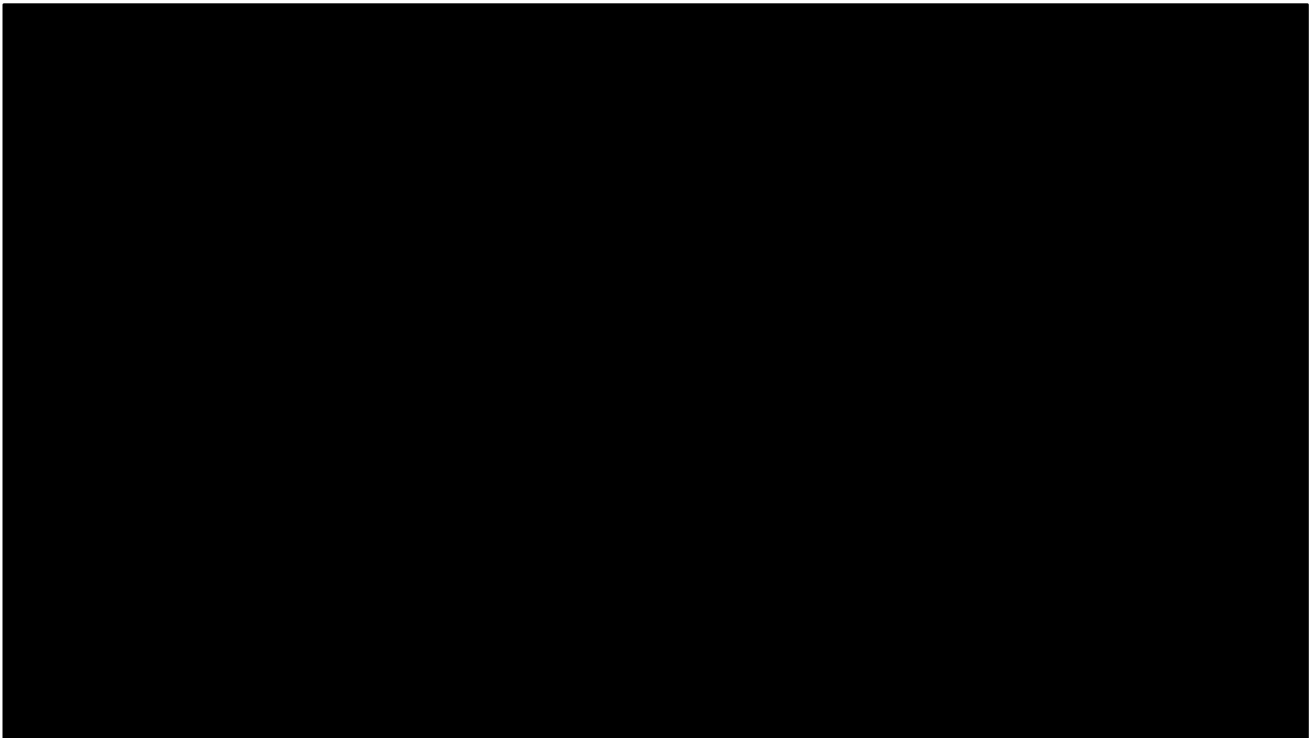
- ShaderFX easy to use
- Everywhere



[Kieran]

For Shaders we decided to use ShaderFX. For those who don't know, ShaderFX is a node based shader graph that comes built into Max and Maya. This was great because we only need to create the shaders once and they're loaded into both packages. And it's super easy for our TDs to keep them up to date with the engine shaders, so we don't depend on availability of 3D programmers.

We just expose parameters in the graph with names that match the engine materials, and the binding is automatized with very simple code.



[Kieran]

Disclaimer: Loading times in DCC are shortened for the video

So let's see it in action!

In this example we select a character in the engine and import her into Max. The Passenger framework mentioned earlier means the game engine and DCC communicate through a connection. So the game engine looks at the selection, sends a list of file names to the DCC which loads them from disk through the pipeline. After making some changes we export one of the meshes back the engine and you can see it instantly hot reload, through the help of a simple windows directory watcher.

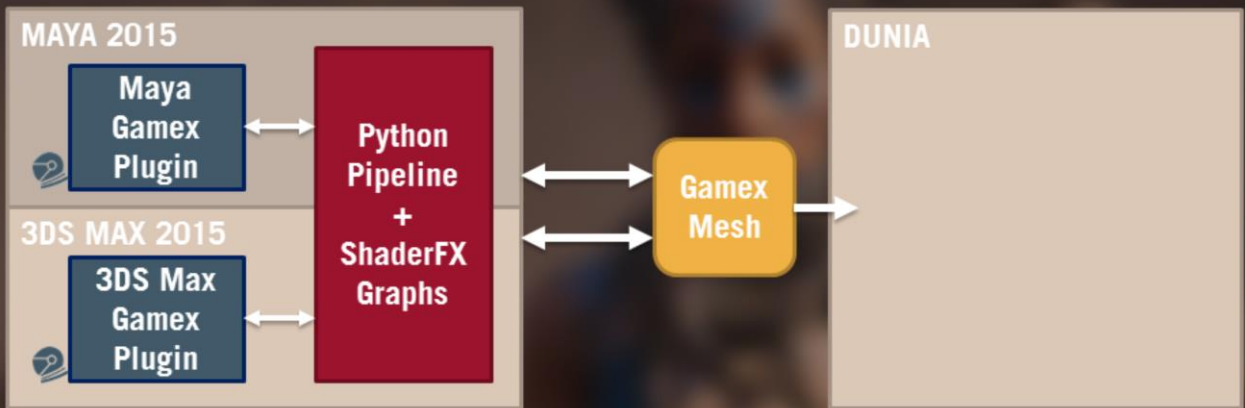
There are little notifications that pop up on the bottom right of the screen showing the file has been checked out of Perforce.

This is also where we put error messages or warnings. We don't block the user or force them to click "Ok" to continue.

You can see if we export multiple items the notifications tack like you expect, so you don't miss an error message.

Just to go full circle then we load the mesh into Maya and change the mesh gain.

GAMEX MODELING PIPELINE



[Julien/Kieran]

So that's the GamEx modelling pipeline.

As you can see, we have one set of shaders and Python tools that share code and work for both packages.

And we have no more bloated source files!

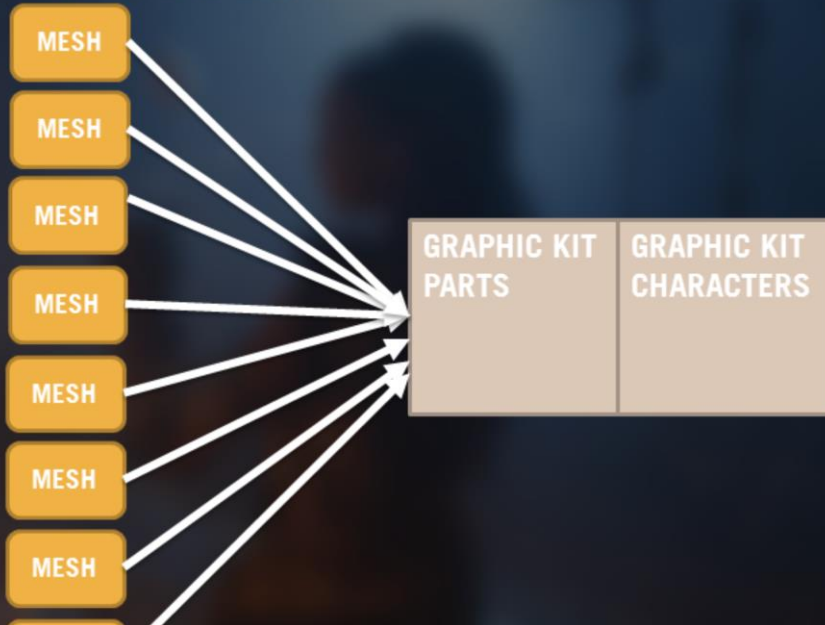
[illegible]

But once the mesh is in the engine we still need a system to put these different meshes together to make a complete character...

It had to go.

FC3/FC4 TOOLS

GRAPHIC KIT



FARCRY
PRIMAL



[Julien]

That's where we turn to Wolfskin our new Character System.

Wolfskin really started as a large scale character customization system but became a complete pipeline.

It aims to remove complex tasks from the daily workflow, allow large collaboration, and eliminate data duplication.

Wolfskin is completely data oriented, so to understand the system, it's a good idea to look at its data. There are two types of data in wolfskin : config files, and item files.

WOLFSKIN CONFIG

DATA DRIVEN

1. SLOT HIERARCHY

```
<FolderSlot slot="Human">
  <FolderSlot slot="Morphology">
    <ItemSlot slot="Head"/>
    <ItemSlot slot="Body"/>
  </FolderSlot>
  ...
</FolderSlot>
```

FARCRY
PRIMAL

[Julien]

First let's look at the config file. The config file is an xml text file that is used by all entities from a group. It describes how an entity is built. For instance here for humans of the player tribe.

There are 3 sections in the config file. The first is the slot hierarchy. It defines which parts or information make the full character. For instance here we have a body, hair, top and bottom clothes, but also a gender, some body paint... We'll come back to those, and the hierarchy later.

WOLFSKIN CONFIG

DATA DRIVEN

2. SOLVER RULES

```
<IfNotEmptyReject  
  slot="Headwear"  
  rejectSlot="Hair"  
  rejectTag="IsThick"/>  
...
```

FARCRY
PRIMAL

[Julien]

The second part of the config files are solver rules. This is a ruleset shared by all entities that target this config file, and describes how to successfully build the character.

Wolfskin has to run a solver on characters to figure out what meshes etc to display, so the rules are for the solver. We'll cover this section in its own chapter.

WOLFSKIN CONFIG

DATA DRIVEN

3. AUTOMATION HINTS

```
<SlotFolderHint  
  defaultFolder="Human\Body"  
  slot="Body"/>  
...
```

FARCRY
PRIMAL

[Julien]

Finally, we have automation hints. These are naming and folder structure conventions that aim to speed up the workflow and reduce number of clicks needed to setup data.

SLOT HIERARCHY

Meshes



[Julien]

- Red slots : typically where we find meshes
- Green slot : top slot, represents the full entity
- Orange slot : grouping nodes within the hierarchy
- Items target slot and associate values in the tree
- Item can set values for their slot or below, including through the use of other items
- Items targeting top slot = full character = DNA

SLOT HIERARCHY

Enums, flags



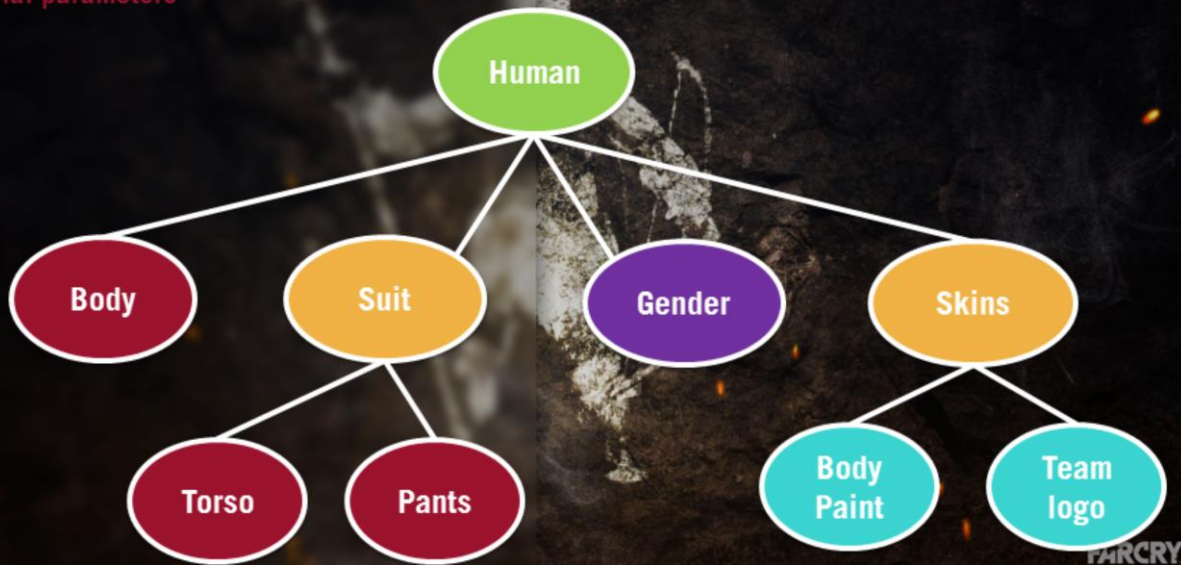
[Julien]

There are other types of data associated with the character, here we've added a gender slot. It's a container for an enumerated value that can take Male or Female as values in our entity group. Note that the enum slot could be anything you want. Wolfskin is data oriented and has no idea what a gender is. It just knows it expects these particular characters to have one of the possible config defined values here.

We'll see what happens with that with the rules.

SLOT HIERARCHY

Material parameters



[Julien]

Finally, we also have material parameter slots in the hierarchy. Here we exposed two textures, possibly colors etc.

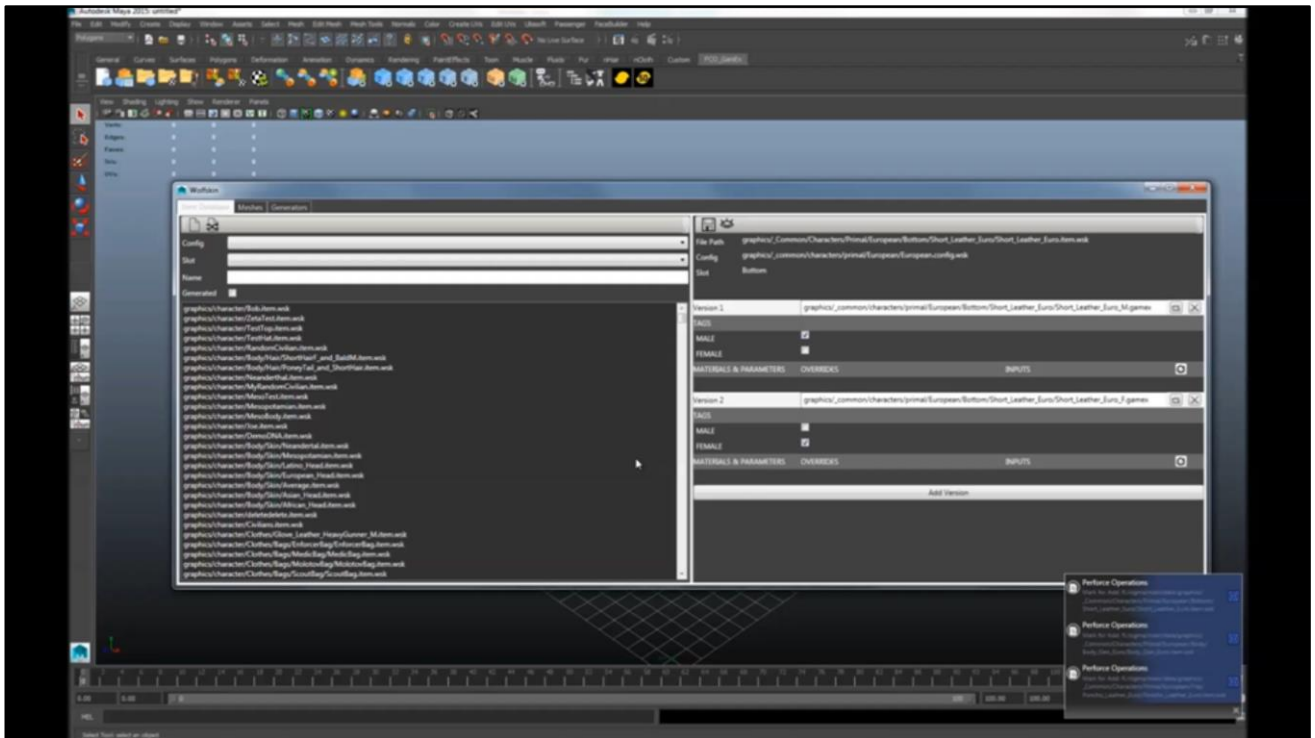
WOLFSKIN ITEM

Human



[Julien]

- Wolfskin item : data that targets a slot of config
- Here, target top slot = full human (we call items targeting the top slot DNAs)
- Provides values for that slot or below
- A DNA does not directly reference specific meshes, but instead wolfskin items for the slots below (which may reference 1 or more meshes when targeting red slots)
- Let's see how we create it



[Julien]

The Wolfskin UI is integrated where artists work, so here we're using Wolfskin in Maya. We have a database view on the left to browse items, and a view of the selected data on the right.

For this demo, we ask Wolfskin to create an item. We target the European tribe config file, and its top slot.

Note: From now on we usually refer to items targeting the top slot of a config file as "DNAs" as they represent the recipe to create a full character.

On the right, you can see that the UI shows pickers for every slots in the hierarchy. This is a rule of the system: a slot in the hierarchy can provide values for any slot below itself. However, in the case of item slots below the slot we're editing, we are not directly picking raw data such as meshes. We are picking existing items that provide these values. Let's pick a body, and some clothes, and look at our character.

Wolfskin can control the modelling pipeline directly, so it asks to load all the gamex meshes necessary. Every piece of data is separate as in a customization system, so I can switch my character's gender and see that all the meshes will reload in the other gender's version.

Wolfskin exports its files as Dunia resources. I will drop a character and point to the wolfskin item I've just created.

Like meshes, wolfskin files are watched by a simple directory watcher that will trigger hot reloading as soon as a file changes. So if I press save in Maya, I will see my character hot reload.

WOLFSKIN ITEM

Human



[Julien]

- That was creating a character
- Let's look at creating items targeting other slots

WOLFSKIN ITEM

Shoes

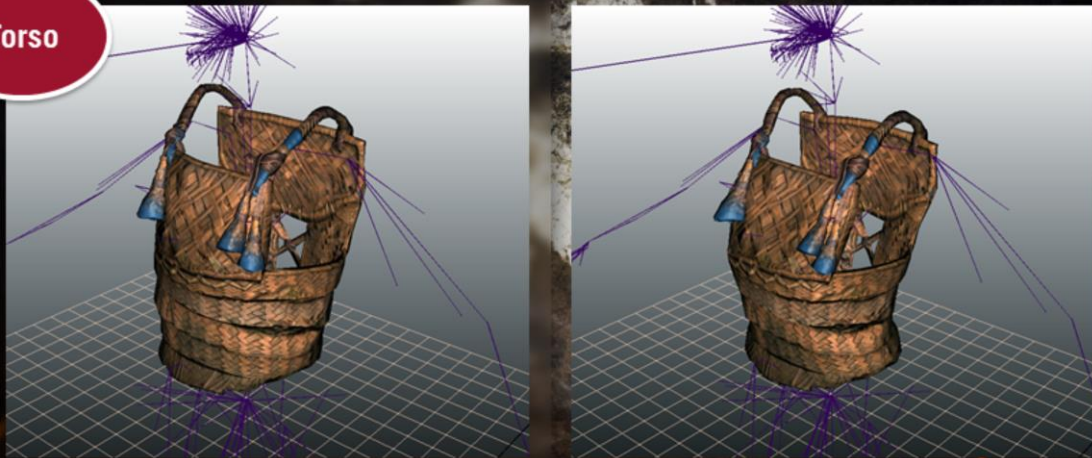


[Julien]

- Shoes is a mesh slot, so that's where we actually chose the mesh.
- Shoes are often unisex in Primal = only 1 mesh

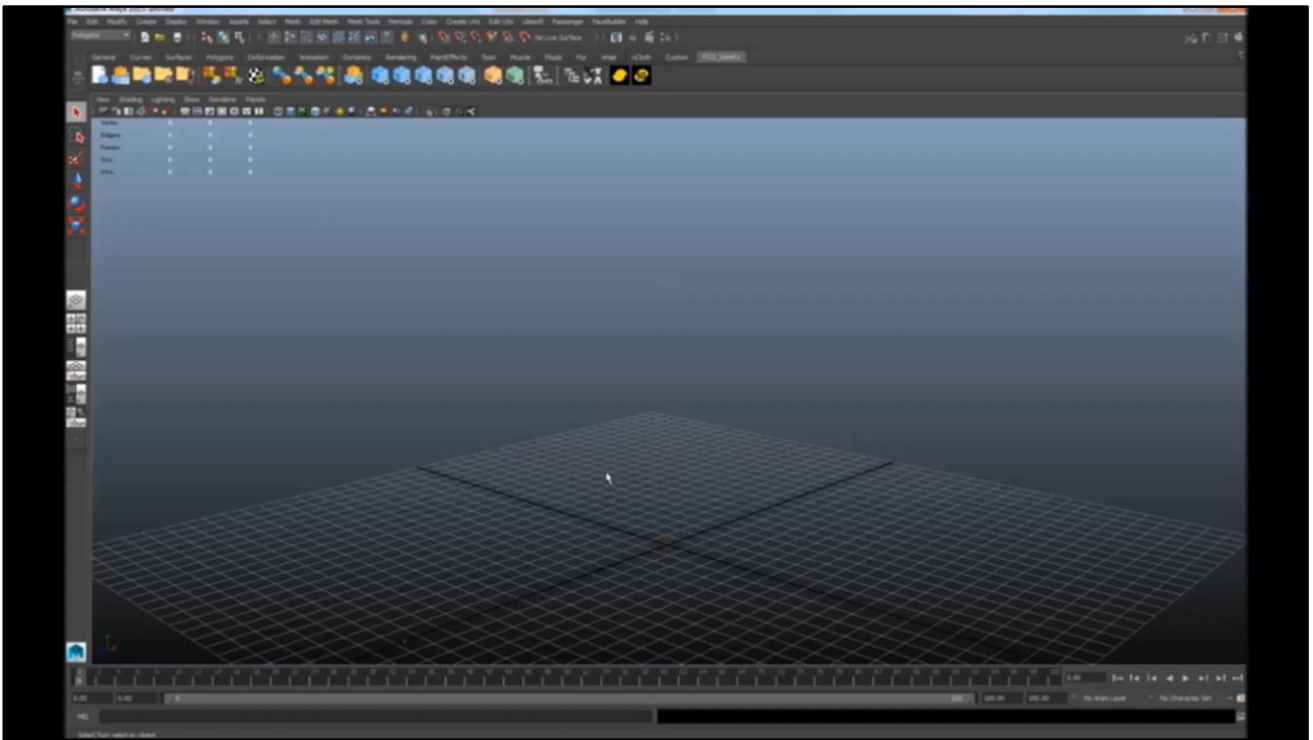
WOLFSKIN ITEM

Torso



[Julien]

Torso items however, have more than one version. Here this izila armor piece has a male and a female version. They were built from the same mesh but tweaked differently, and exported as two separate assets, then referenced together in a wolfskin item.



[Kieran]

Let's see how we create a usable wolfskin item out of these two meshes. First let's look at our meshes.

You can remember that the config file had some automation rules. One of these says that meshes found in the folder named Top are probably meshes for the Top slot of the Izila tribe config file. So I can go in wolfskin, simply select those two meshes, and the system will guess that I'm trying to create a Top for this tribe and preselect everything. If it's correct, I can click ok. It's now gonna guess a name for the item based on the similarity between the names of the two meshes. In my case I have the same name underscore M for male, and underscore F for female.

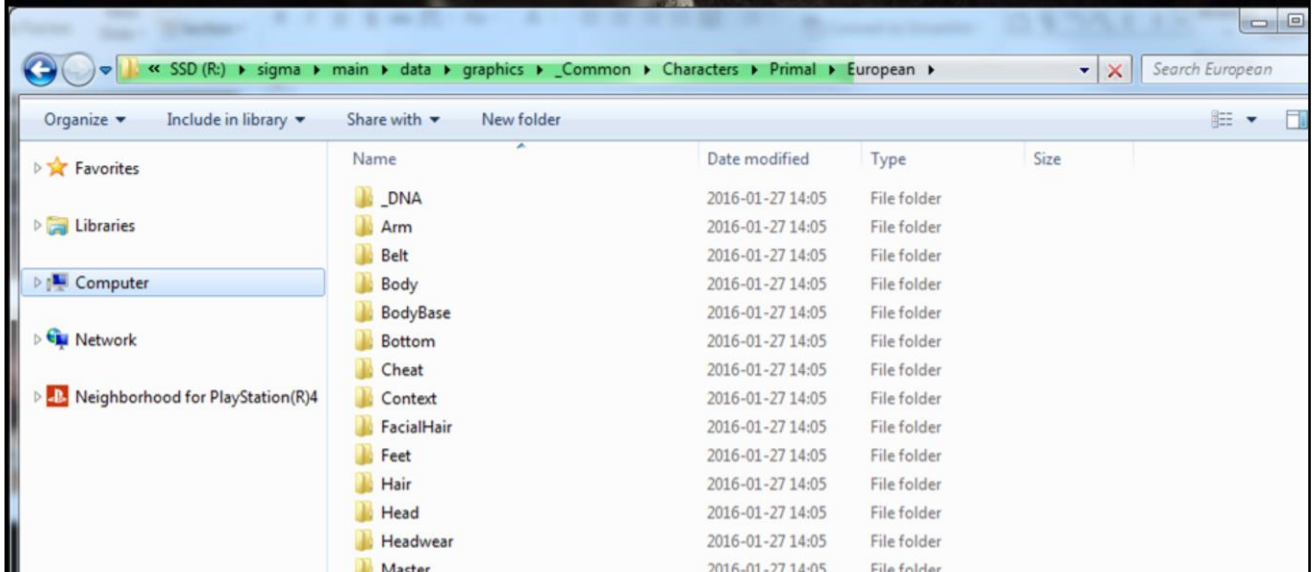
The last part of the automation hint actually interprets those _M and _F to mean that the meshes should be automatically tagged Male and Female (or _MF for unisex items). Once again I can just check that the result is fine, and that's it. My item is usable in any DNA and the right mesh will show up thanks to the rules and solver.

As I said, wolfskin has no hardcoded knowledge of gender or anything, so the check boxes on the right here with the available tags are deduced from the config file rules. Because there's a gender rule that expects meshes to be tagged male and or female in all the slots, appropriate checkboxes are added.

For the sake of the demo we create other items in the same way.

AUTOMATION HINTS

Folder structure to slot mapping



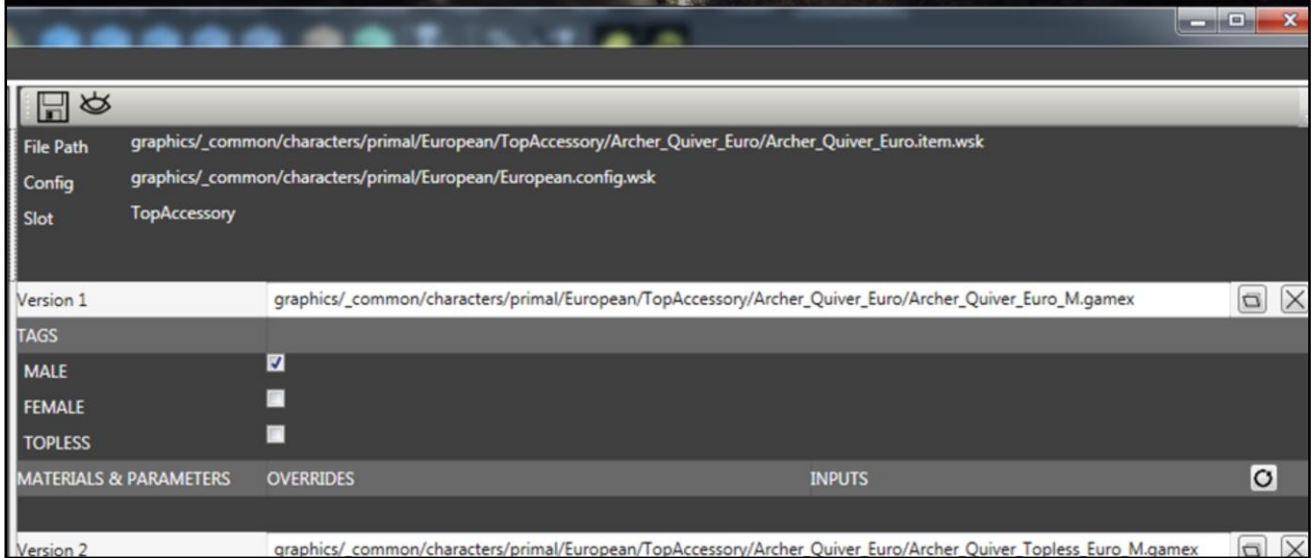
[Julien]

To recap: 2 types of automation hints

- 1 the folder structure to slot mapping. In the wolfskin config, we can specify that a certain folder is probably of a certain config/slot.

AUTOMATION HINTS

Naming convention to tags



[Julien]

2 : the asset suffixes to tags convention. In the config file, we can ask wolfskin to automatically tag MALE any mesh that has the suffix _M, or MALE+FEMALE for any item that has the suffix _MF and so on.

The automation hints help to enforce clean data structure and naming conventions, and make asset integration really fast.



[Julien]

Now we've seen that the multiple versions of the meshes in an item are tagged.

Let's take a moment to see how the wolfskin solver selects the right meshes to display in a DNA.

RULES

Solving characters

Select the best versions



```
1 <WolfskinConfig>
2   <Rules>
3     <RequireEnumTag      slot="Gender"/>
4
5     <Bonus                score="1"
6                           slot="TopAccessory"
7                           tag="Topless"/>
8
9     <Bonus                score="1"
10                           slot="Haircut"
11                           tag="IsThick"/>
12
13     <IfNotEmptyReject    slot="Top"
14                           rejectSlot=
15                           "TopAccessory"
16                           rejectTag="Topless"/>
17
18     <IfNotEmptyReject    slot="Headwear"
19                           rejectSlot="Haircut"
20                           rejectTag="IsThick"/>
21
22     <IfQualifierReject   slot="Top"
23                           tag="CoversTheHead"
24                           rejectSlot="Haircut"
25                           rejectTag="IsThick"/>
26
27     <IfQualifierReject   slot="Outerwear"
28                           tag="CoversTheHead"
29                           rejectSlot="Haircut"
30                           rejectTag="IsThick"/>
31   </Rules>
32 </WolfskinConfig>
```

- The rules of the config file are run on any character DNA before it can be displayed. They have to be run each time anything changes in the DNA too as items can affect each other through the rules. Example of use for the rules are:
 - Selecting the right gender versions
 - Solving conflicts between meshes (hats collision with hair etc)
 - Adapting meshes to other clothes (bqg closer to body when not wearing a coat)
 - User preference in a customization system by using a Boolean slot for instance (uncover head when wearing a hoodie)

SOLVER EXAMPLE

DNA A

Rules

- Require tag from enum slot “Gender”

FARCRY
PRIMAL

[Julien]

Let's look at a first simple example.

We have one rule in our config file, the gender rule. It requires any selectable mesh to carry the tag found as value for the enum slot “Gender”.

We create an empty DNA.

SOLVER EXAMPLE

DNA A

Body A
Versions
Male
Female

Leather Top
Versions
Male
Female

Pants
Versions
Male, Female

Rules

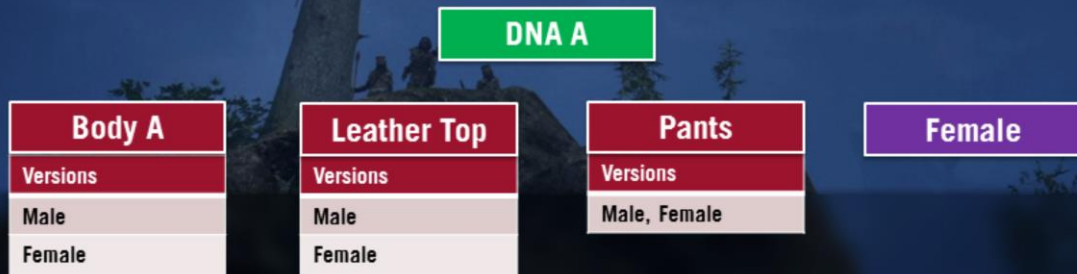
- Require tag from enum slot "Gender"

FARCRY
PRIMAL

[Julien]

We select a few existing items, the body and the top have a male and female version but the pants is unisex so it has both tags on the same mesh.

SOLVER EXAMPLE



Rules

- Require tag from enum slot "Gender"

FARCRY
PRIMAL

[Julien]

We select the value Female for the Gender enum slot. We now have a complete character and are ready to run the solver.

SOLVER EXAMPLE

DNA A

Body A	
Versions	Score
Male	0
Female	0

Leather Top	
Versions	Score
Male	0
Female	0

Pants	
Versions	Score
M, F	0

Female

Rules

- Require tag from enum slot "Gender"

FARCRY
PRIMAL

[Julien]

The solver starts by initializing scores for each mesh at 0 point.

SOLVER EXAMPLE

DNA A

Body A	
Versions	Score
Male	X
Female	0

Leather Top	
Versions	Score
Male	X
Female	0

Pants	
Versions	Score
M, F	0

Female

Rules

- Require tag from enum slot "Gender"

FARCRY
PRIMAL

[Julien]

We then run each rules in order.

Here the "Require tag from enum" rule demands that all meshes are tagged by the value of the enum Gender. All Male mesh assets become unselectable and are rejected.

SOLVER EXAMPLE

DNA A

Body A	
Versions	Score
Male	X
Female	0

Leather Top	
Versions	Score
Male	X
Female	0

Pants	
Versions	Score
M, F	0

Female

Rules

- Require tag from enum slot "Gender"

FARCRY
PRIMAL

[Julien]

Solving is over, there is only one available mesh in each item. These will be displayed.

If Wolfskin ends the solving phase with more than 1 mesh with the best selectable score, it will produce an error and refuse to display the mesh. This is because it means we have an unsolvable conflict in the data. The item or the rules should be fixed.

Having 1 or 0 mesh with highest selectable score is a valid result.



[Julien]

Let's look at a more complex example. In this video, we're going to add a prehistoric "hoodie" to the character. It's a torso item that includes a hood cover the head of the character. Now pay attention to that haircut.

Notice how when I put the hood, the back of the haircut that would overlap disappears, but the recognizable front fringe of the character is still visible. To make this possible we created an alternate version of that same haircut where we removed the back part.

Now what happens if I want to add a big hat ? Because this relies on a customization system, we want to handle every piece individually but still have correct visual. Notice that when I put the hat, the hood of the hoodie falls back. As nothing is blocking the haircut's back anymore, it comes back.

SOLVER EXAMPLE 2

Rules

- **Require tag from enum slot “Gender”**
- **Bonus CoversTheHead in Top : + 1**
- **If Hat IsThick reject CoversTheHead in Top**

FARCRY
PRIMAL

[Julien]

Let's look at how you'd setup a config file's rules to deal with the hooded tops versus hats scenario.

We add two rules to the config file. The first one is the bonus rule. It declares the defaults situation. By default, versions of meshes that have the CoversTheHead tag in Top have a bonus of 1 point. This ensures that when both CoversTheHead and non CoversTheHead versions are available, we select the right mesh.

Finally, we add a conditional rule for the solver : if hat isthick, reject coversthehead in top. This conditional rule explicitly states that thick hats have priority over hoods.

SOLVER EXAMPLE 2

Top: Hoodie	
Versions	Score
Male, CoversTheHead	0
Female, CoversTheHead	0
Male	0
Female	0

Hat: Fancy Top Hat	
Versions	Score
Male, IsThick	0
Female, IsThick	0

Rules

- **Require tag from enum slot “Gender”**
- **Bonus CoversTheHead in Top : + 1**
- **If Hat IsThick reject CoversTheHead in Top**

FARCRY
PRIMAL

[Julien]

If you look at our hoodie, you see it has four meshes. One for each gender that covers the head, and one for each gender that doesn't. Notice that our hat only has two versions, each with an extra tag that says the hat IsThick. This is to specify that this hat would not fit under a hood.

SOLVER EXAMPLE 2

Top: Hoodie	
Versions	Score
Male, CoversTheHead	X
Female, CoversTheHead	0
Male	X
Female	0

Hat: Fancy Top Hat	
Versions	Score
Male, IsThick	X
Female, IsThick	0

Rules

- **Require tag from enum slot "Gender"**
- **Bonus CoversTheHead in Top : + 1**
- **If Hat IsThick reject CoversTheHead in Top**

FARCRY
PRIMAL

[Julien]

First the gender rule eliminates half of the meshes.

SOLVER EXAMPLE 2

Top: Hoodie	
Versions	Score
Male, CoversTheHead	X
Female, CoversTheHead	0 +1 = 1
Male	X
Female	0

Hat: Fancy Top Hat	
Versions	Score
Male, IsThick	X
Female, IsThick	0

Rules

- Require tag from enum slot "Gender"
- **Bonus CoversTheHead in Top : + 1**
- **If Hat IsThick reject CoversTheHead in Top**

FARCRY
PRIMAL

[Julien]

Then, the default rule favors tops that cover the head. If the hat wasn't marked thick solving could end here but...

SOLVER EXAMPLE 2

Top: Hoodie	
Versions	Score
Male, CoversTheHead	X
Female, CoversTheHead	X
Male	X
Female	0

Hat: Fancy Top Hat	
Versions	Score
Male, IsThick	X
Female, IsThick	0

Rules

- Require tag from enum slot "Gender"
- Bonus CoversTheHead in Top : + 1
- If Hat IsThick reject CoversTheHead in Top

FARCRY
PRIMAL

[Julien]

The conditional rule detects there is at least one selectable mesh with the IsThick tag in the hat, so the top marked CoversThehead is made unselectable.

SOLVER EXAMPLE 2

Top: Hoodie	
Versions	Score
Male, CoversTheHead	X
Female, CoversTheHead	X
Male	X
Female	0

Hat: Fancy Top Hat	
Versions	Score
Male, IsThick	X
Female, IsThick	0

Rules

- Require tag from enum slot “Gender”
- Bonus CoversTheHead in Top : + 1
- If Hat IsThick reject CoversTheHead in Top

FARCRY
PRIMAL

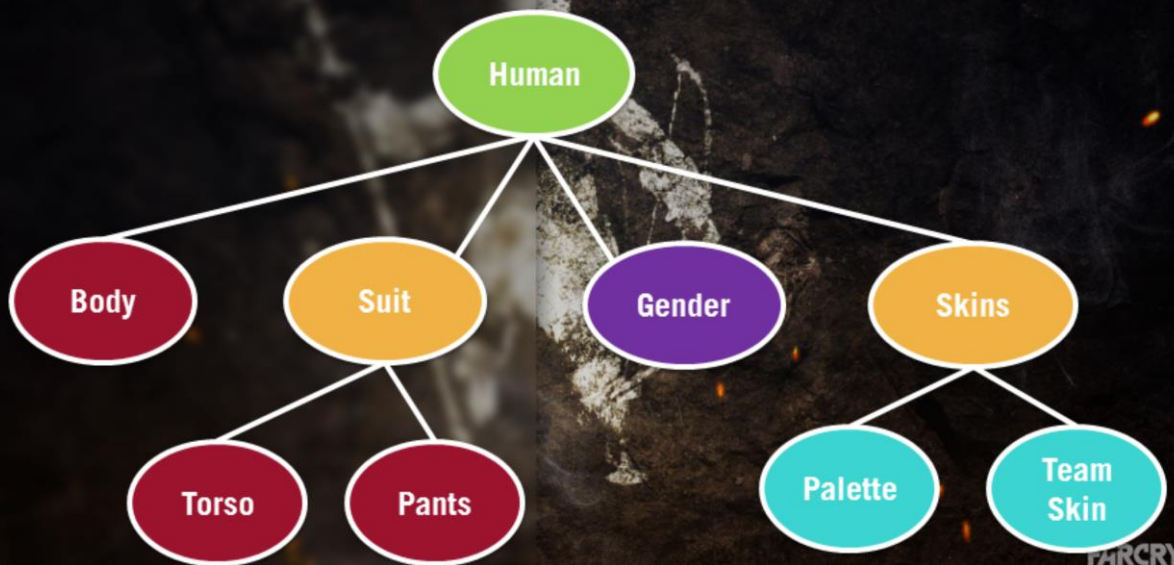
[Julien]

The solver has successfully selected which meshes to display.

Now the important thing to note here, is that yes you have to think a bit when setting up the rules in the config file. But once this is done, each time you add items you only have to describe the new content. During production, when you create items, you never have to think about all possible combinations, you simply describe meshes, “Does this top covers the head ? Is this hat thick ? ” and provide enough versions in case you have things that can be excluded.

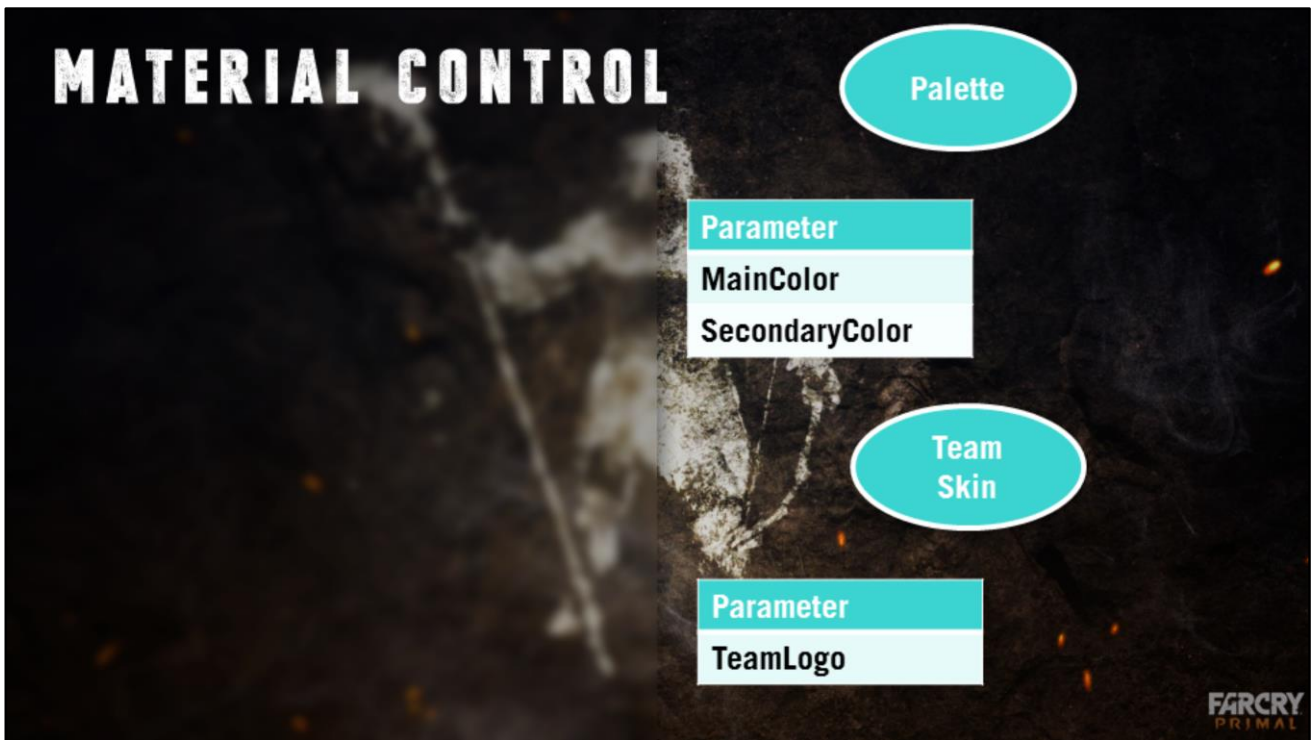
Because all rules are in data, and we never think about everything at the beginning of the project, it’s totally ok that the rules will change during production. You don’t have to show up at the last minute and beg a programmer to change the logic, the ruleset can be changed locally and fixed with a data submit. We actually did that several times through primal, at one point we added special versions of arrow quivers for topless characters, so that they better rested close to their bodies when they had no shirt on. The amount of data is manageable, and setting up the rules was done in a few minutes.

SLOT HIERARCHY



[Julien]

- Now you have seen the mesh slots
- And you hopefully understand the rules and solvers
- What about other slot types ?
- We've seen Gender is a simple Enum slot
- Let's look at material parameters and how they interact with the character

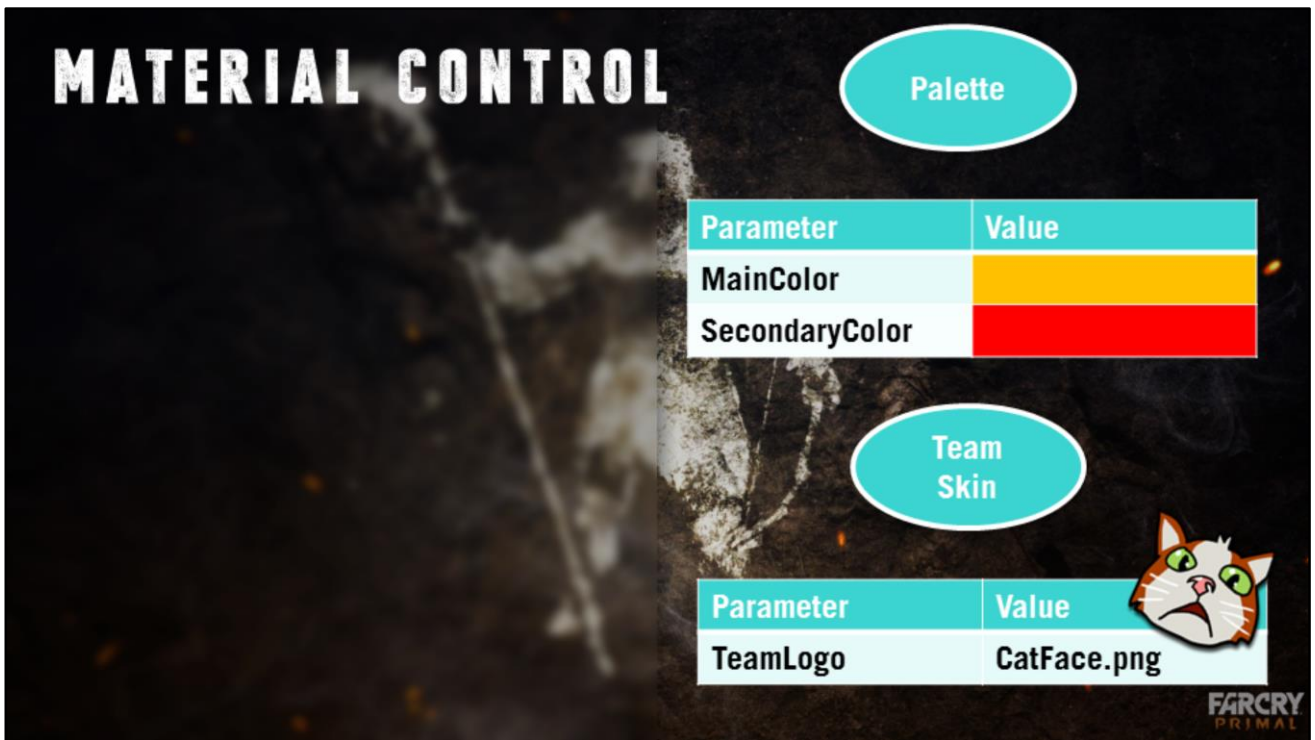


[Julien]

When declaring material parameter slots, you have to declare a list of typed parameters.

For instance here we declare the palette as two colors named MainColor and SecondaryColor. Likewise, we declare the TeamSkin slot as a container for a texture parameter called TeamLogo.

Note that at this point we define no logic as to what to do with those parameters.

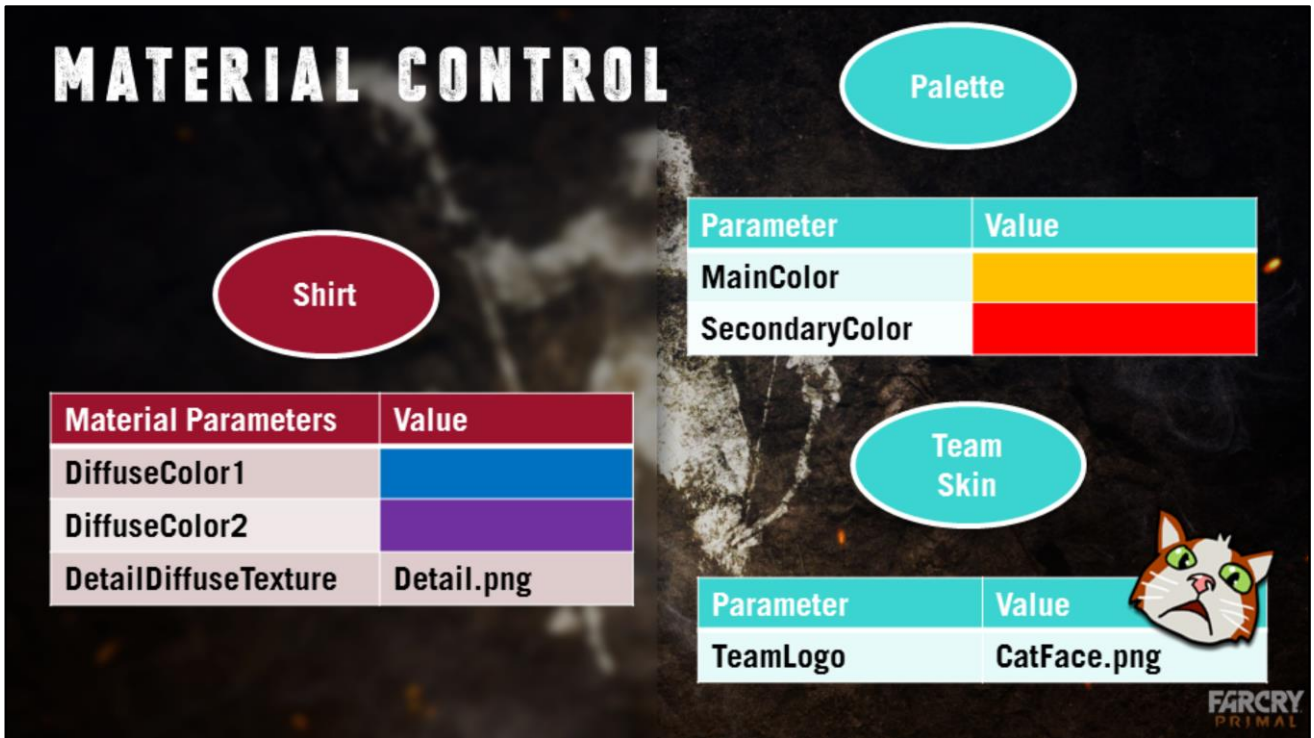


[Julien]

When creating items that target those slots, Wolfskin will let you pick values for the typed parameters. Here we select colors and textures and create our items.

It is therefore really fast to create a lot of those items.

*Note that material parameters also contain versions that can be tagged, and are subject to the solver. This means that you could have material parameters adapt to other settings in the material. For instance, if you had some ethnicity information in the DNA, you could adapt makeup colors to better support every skin tones.



[Julien]

On the other hand, when you create mesh items, Wolfskin lets you discover materials used by the mesh and override any shader parameters for them.

It is possible to set those value by hand, for instance here one of the versions of the shirt mesh override colors and a texture.

MATERIAL CONTROL

Shirt


Material Parameters	Link
DiffuseColor1	MainColor
DiffuseColor2	SecondaryColor
DetailDiffuseTexture	TeamLogo


Palette

Parameter	Value
MainColor	
SecondaryColor	

Team Skin

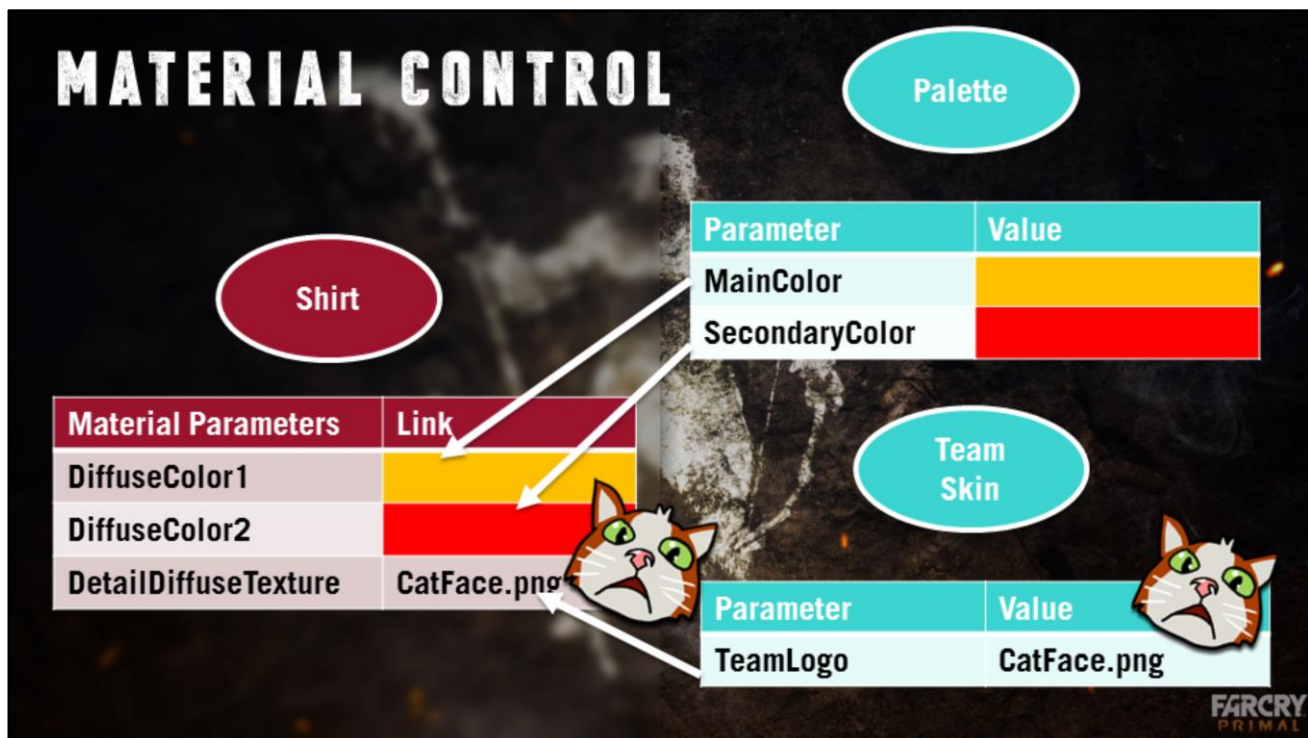
Parameter	Value
TeamLogo	CatFace.png





[Julien]

However, it is also possible through the UI to select a named parameter from the DNA as the source for the parameter override.



[Julien]

In that case, values are name matched when the character is built.

This strategy supports many heterogenous shader setups on the meshes, while at the same time making it really fast to add lots of palettes and material skins. With items owning the logic of how to apply skins, we can manage a very wide range of setups.

In Primal, this system is used to assign class based body paint to the enemies, as well as unlockable downloadable content which was released as pre order bonus (Takkar's arm tattoo).

SLOT HIERARCHY

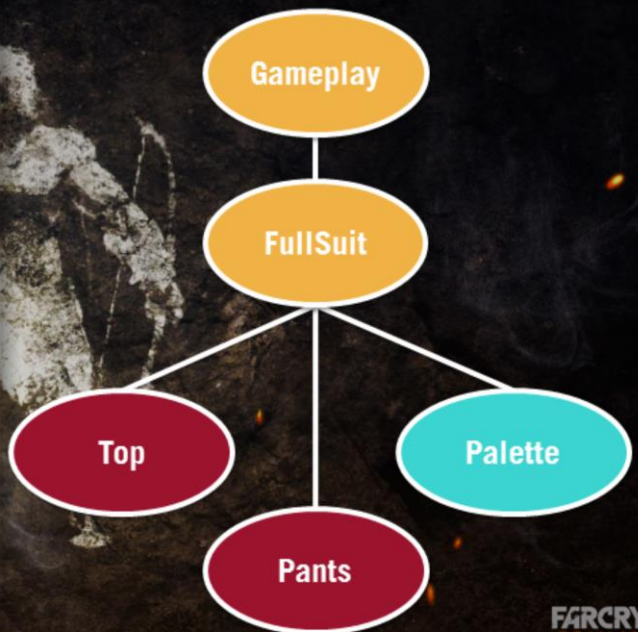


[Julien]

- What about orange slots : Folders

SLOT HIERARCHY

PRIORITIES WITHIN THE HIERARCHY



[Julien]

You'll remember items can provide values for slots at their target level or below.

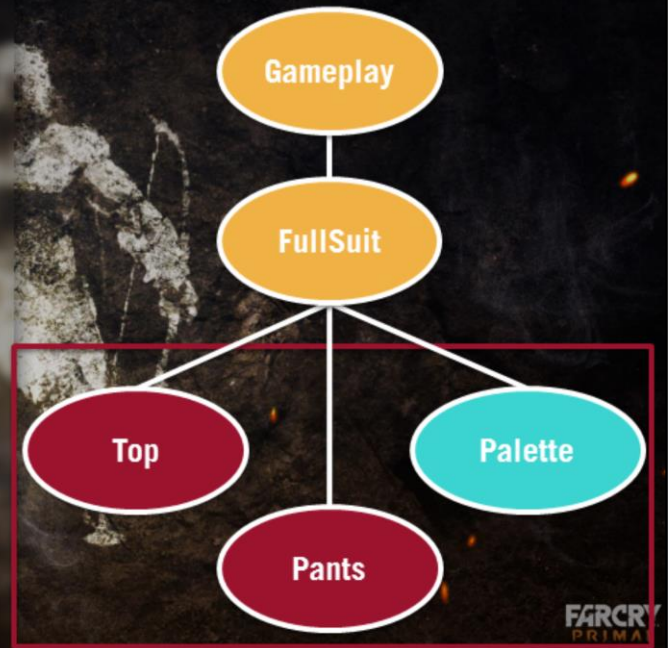
For this example we use a hierarchy that could have been used on an online project like FC4. We have at the bottom two mesh slots and a material parameter slot. These are grouped below a FullSuit folder slot, which is itself below a Gameplay folder slot.

In the DNA, there could be a Top item, but also a FullSuit item which contains a Top, and a Gameplay Item which also contains a Top. So there would be 3 tops in the DNA.

The rule for solving that conflict, is that the higher the slot is, the bigger its priority. Let's see an example.

SLOT HIERARCHY

PRIORITIES WITHIN THE HIERARCHY

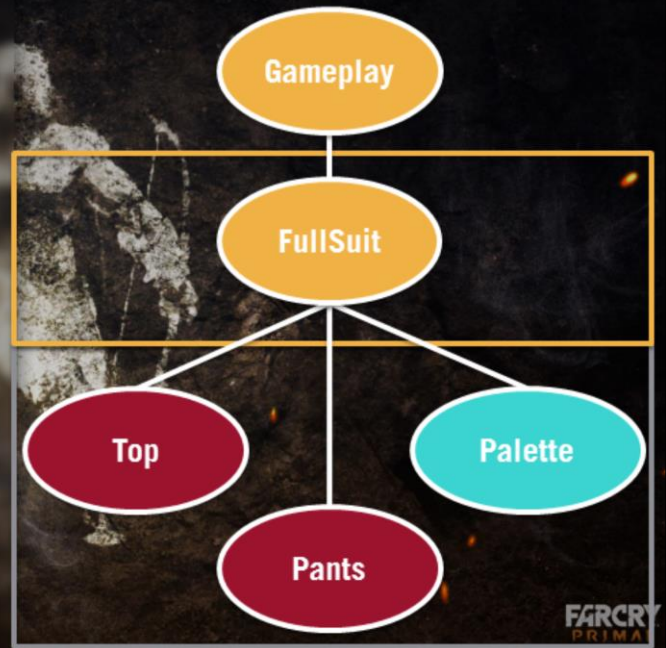


[Julien]

Here a user has selected one top, some pants and maybe a palette.

SLOT HIERARCHY

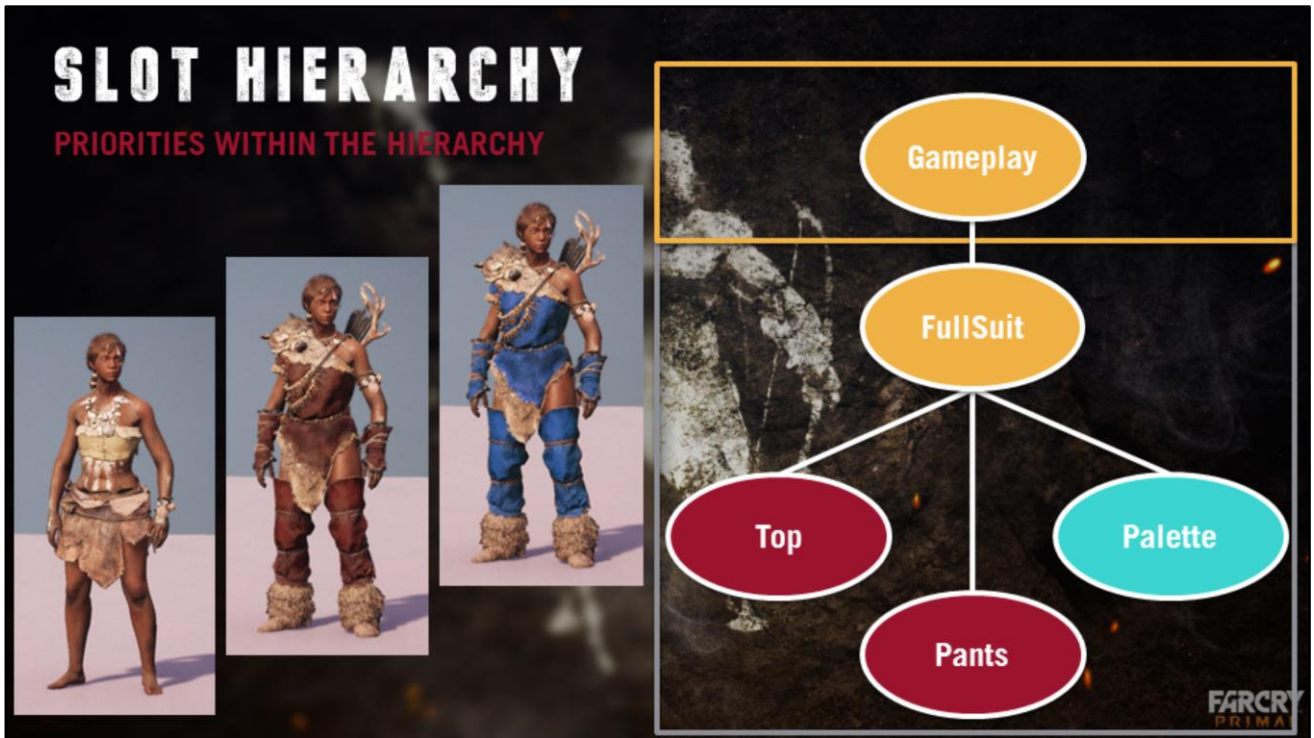
PRIORITIES WITHIN THE HIERARCHY



[Julien]

For some items however, you don't want the user to mix and match. So here we have created a full hunter suit item, which contains pre-selected values for the 3 child slots.

As FullSuit is higher than Top, Pants and Palette, equipping the hunter suit into the DNA causes those meshes to be overridden.



[Julien]

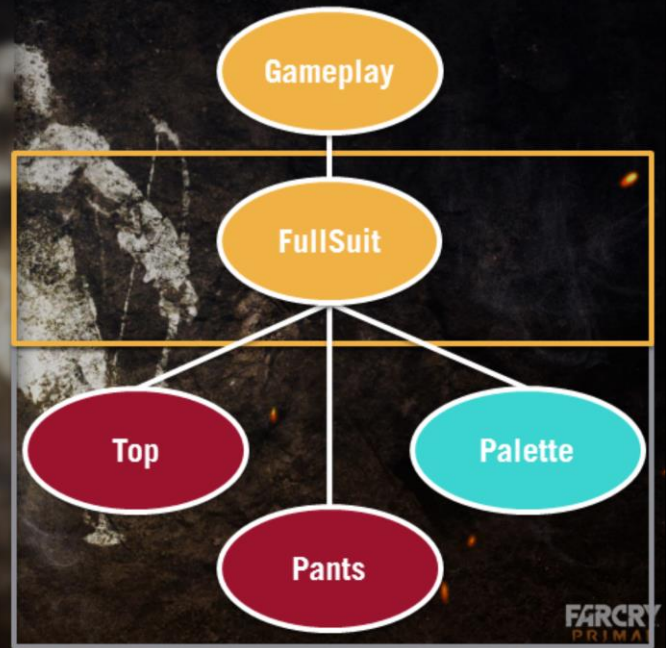
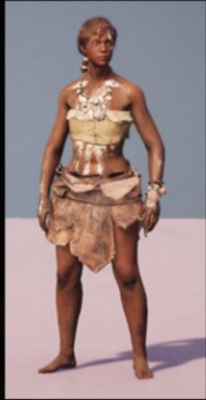
On FC4, we had team based PVP Arena. In such situation, the gameplay team may want to have a way to override the color of the player's outfit to display their team.

Wolfskin allows a simple way to do this by adding a slot in the hierarchy which implicitly handles the conflict. Here, the Gameplay team is given priority over the suit and its child slots.

Therefore, they can create an item that overrides the player palette and apply it to their outfit.

SLOT HIERARCHY

PRIORITIES WITHIN THE HIERARCHY

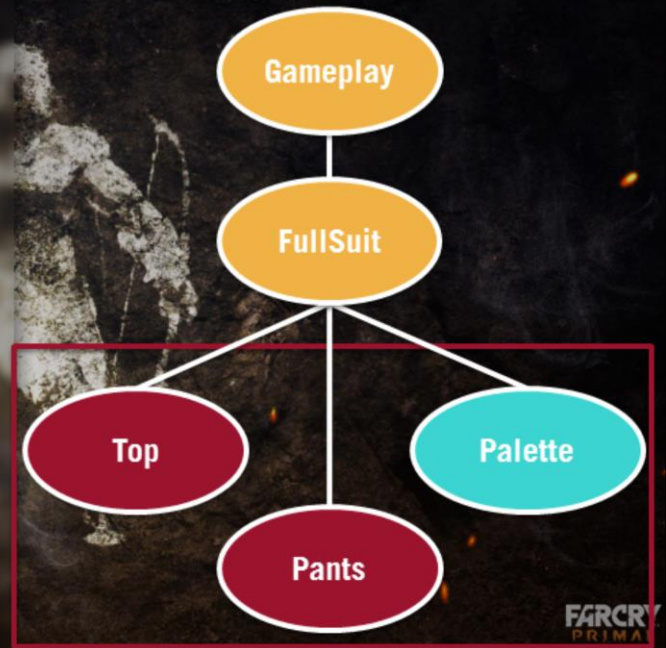


[Julien]

Now because we have 3 levels of items at the same time in the hierarchy, it means removing one of those is also a non destructive operation. As the player exits the arena, the Gameplay team clears its slot, and the player finds their avatar back to their previous state.

SLOT HIERARCHY

PRIORITIES WITHIN THE HIERARCHY



[Julien]

Likewise, the character does not become naked when removing the hunter suit, but reverts to their previous outfit with no extra logic.



[Kieran]

So next up, we're going to talk about variety.



[Kieran]

Basically, how do we go from a large amount of individual assets and combine them in different ways to get...

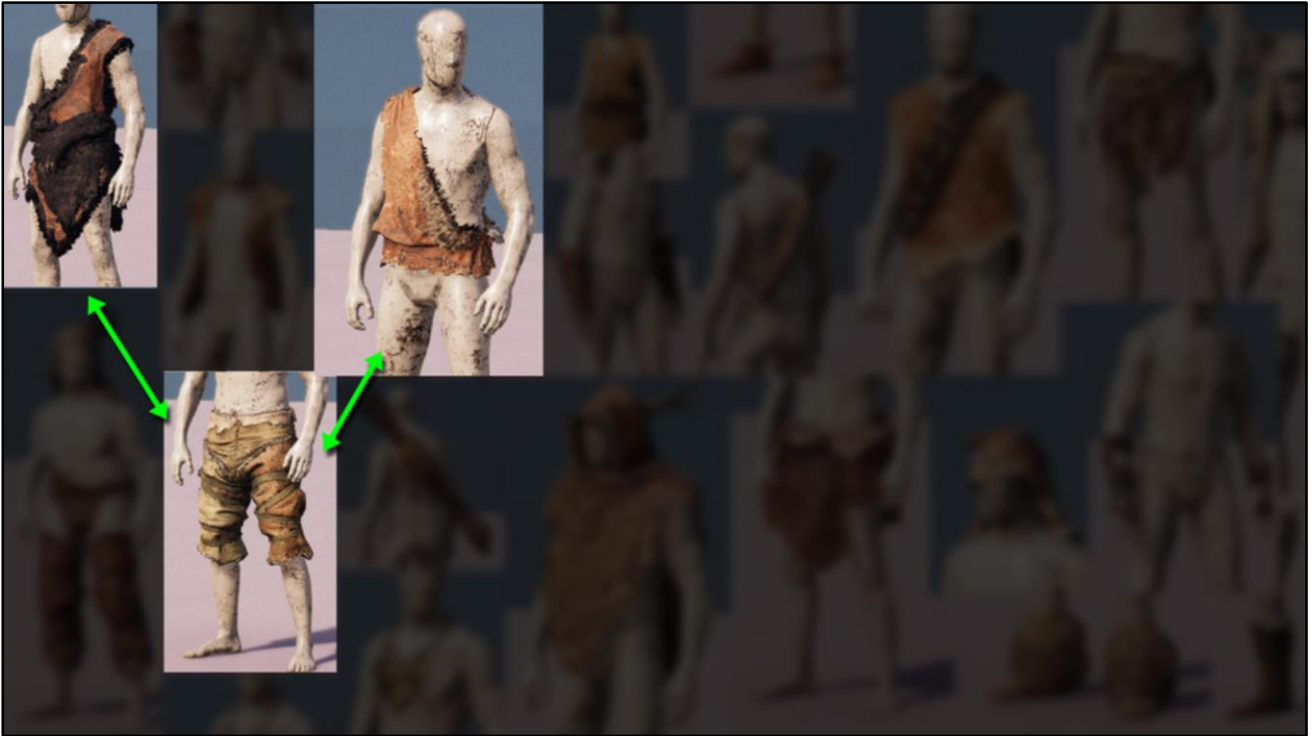


[Kieran]

...an even larger amount of characters!

For Primal we chose to solve this in a very different way than we did in the past!

The first decision we made was that rather than randomly assemble characters from our database of parts, we would create full DNAs of each character in the game.



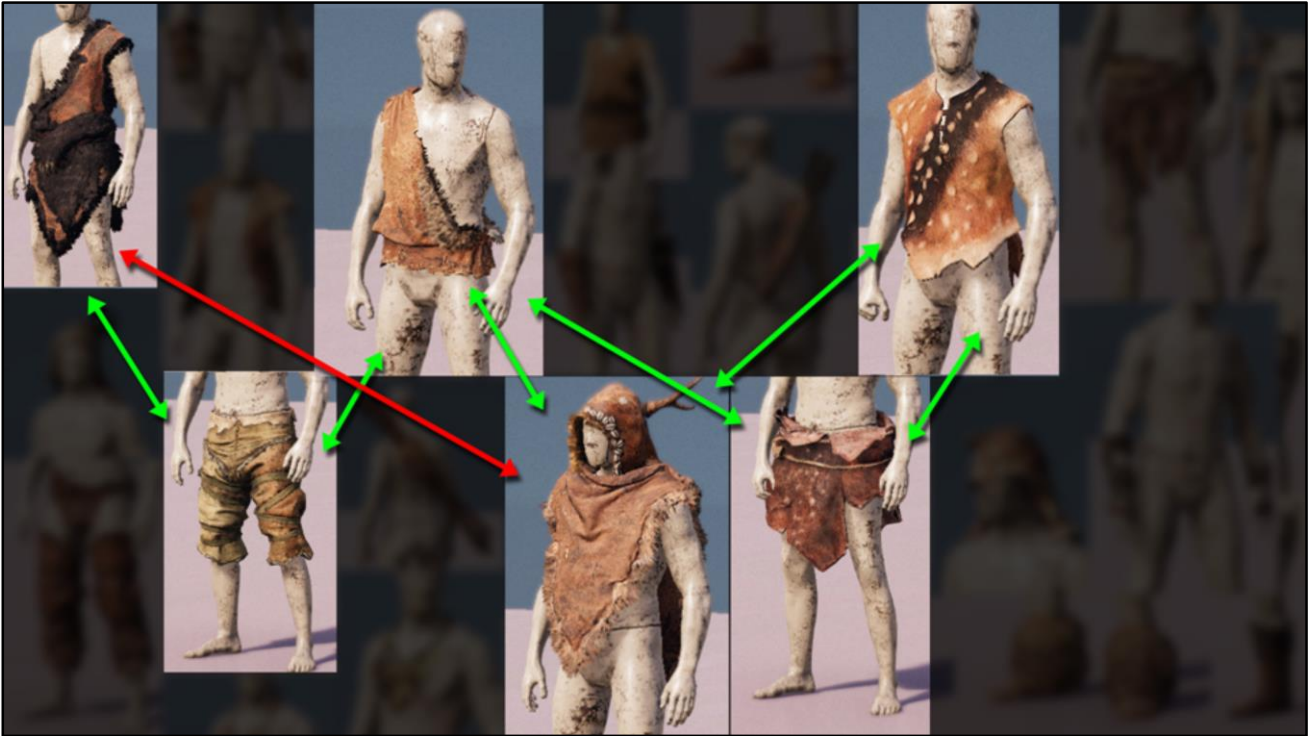
[Kieran]

Let's take for example this pair of pants. So in the past we'd randomly pick different meshes, and set up tags to define which ones go well together.

I've decided that I think these pants goes well with this tunic and shirt.



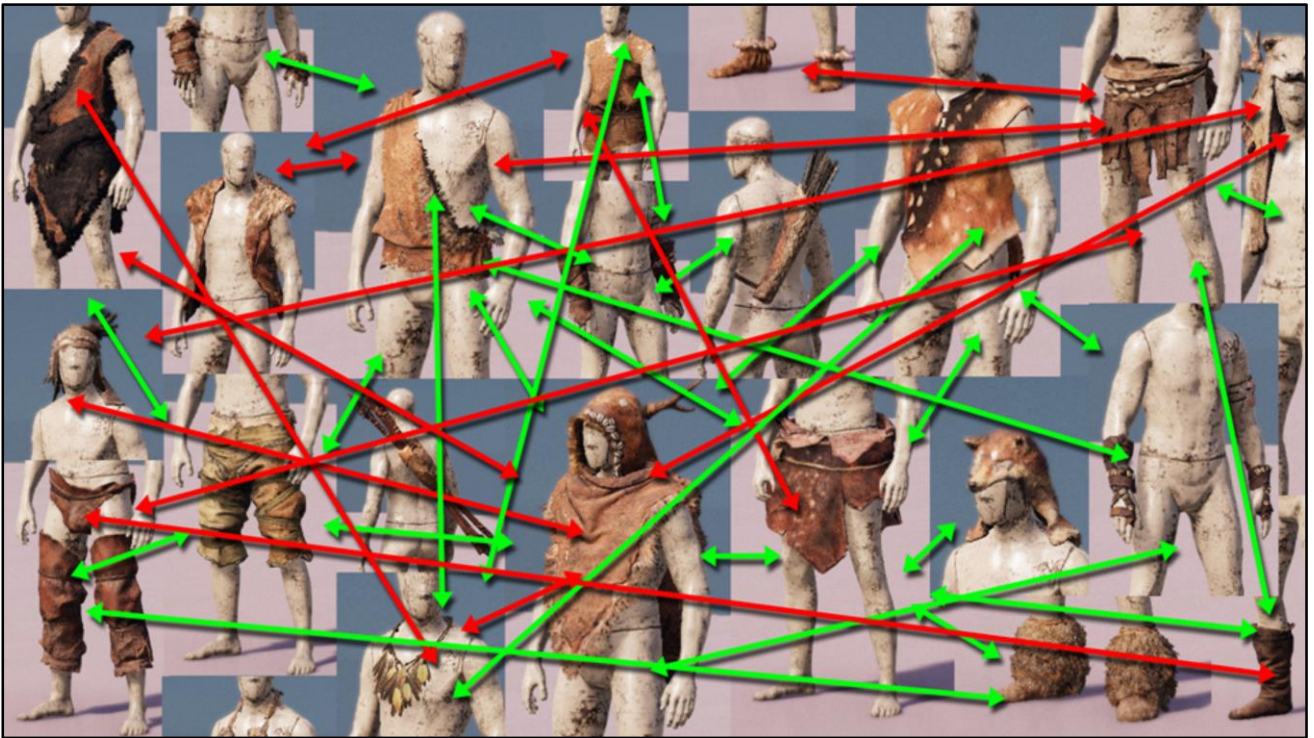
[Kieran]
And this loin cloth goes well with these two shirts.



[Kieran]

Now I'm going to add a coat on top. The coat goes well with some of the shirts BUT it totally breaks if I try to use it with the tunic on the left. I definitely don't ever want these to be put together.

So if I keep setting up these links with every piece of clothing it spirals out of control pretty fast...



[Kieran]

And if these parts are all assembled randomly then I need to set up a **lot** of tags, and I still won't see every possible combination.

Experience has shown us that once our game gets into the hands of millions of players, who each see thousands or hundreds of thousands of combinations they will frequently find things that our artists never saw.

And even when our testers **do** find bad combinations, it can be really hard to track down and identify why it happened, and how to fix it.



[Kieran]

Instead for Primal we break down this complex problem into smaller more manageable problems.

So for example I will take some clothing and put them together into a “Generator”, avoiding any items that don’t go well together, and tell Wolfskin to build me some random DNAs. Wolfskin will tell me I’m building, for example, 6 out of a possible 600 combinations.



[Kieran]

Then I'll pick some other clothing in another Generator and build some more combinations.

And I'll just keep generating groups of DNAs. And I'll add them into lists of DNAs so when we spawn a villager we can randomly chose a character from a list of approved DNAs.



[Kieran]

Then maybe I'll build some DNAs by hand.

If we need a specific quest giver we might find a cool DNA from our list and pull it out so that can be a unique character.



[Kieran]

When I'm finished I have a list of pre-determined DNAs, that will always work correctly. Or more likely I have multiple lists of different types of characters. Villagers, warriors, archers, etc.

If there's a problem with any of the DNAs I can find it and fix it, or I can just remove it from the list.

If I feel like there's not enough combinations I can always go back to my generators and automatically generate some more combinations, or build some good looking combinations by hand. It's totally up to the artists how random or controlled they want to be.



[Kieran]

Let's do a little post mortem of the development

WHAT WENT RIGHT

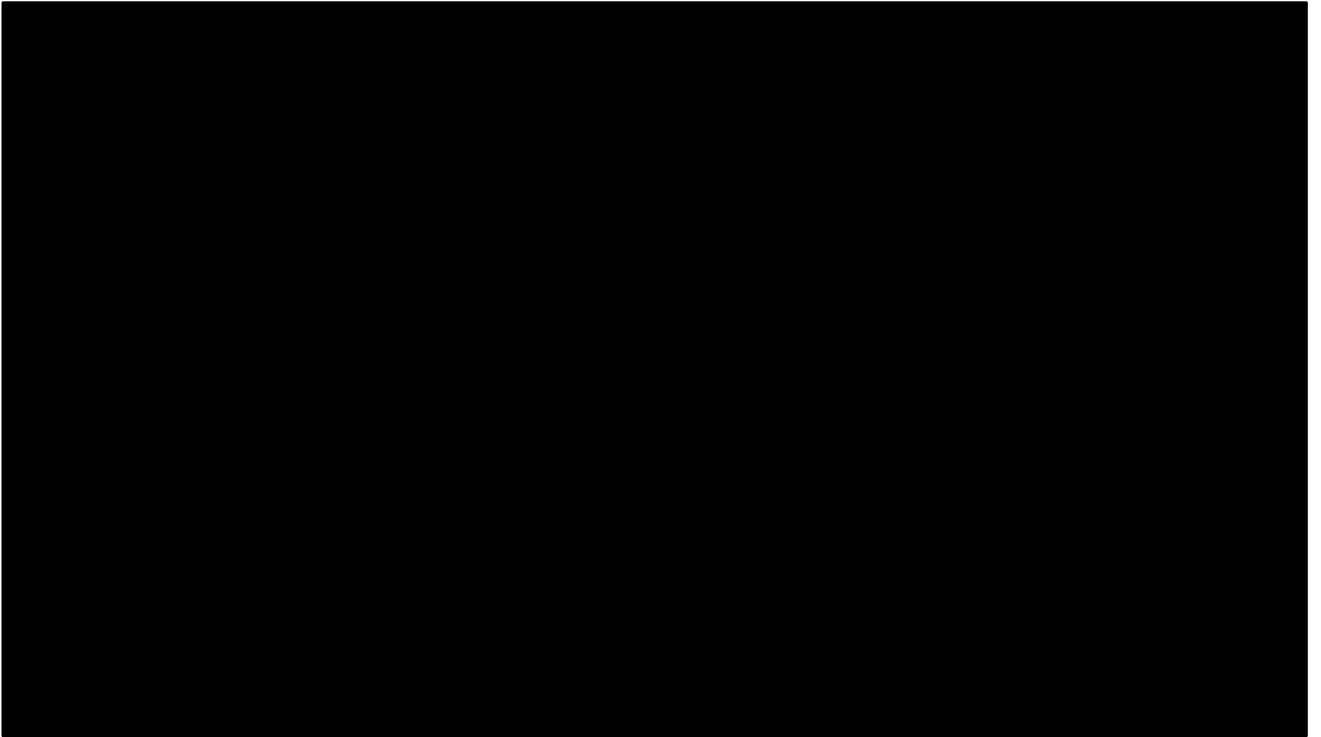
- Complete new character pipeline
- Supported:
 - Skinned characters
 - Skeletons
 - Static and breakable props
- Max/Maya support
- Fast Wolfskin/Gamex iteration loop
- Added DLC and crafting costumes



[Kieran]

List of things that went right,

Let's look at some of those + a bit of final quality characters in the shipped game. Some of those are beyond the scope of this presentation and involve many tech teams at ubisoft, and of course many programmers from the team.



Primal Characters Showreel

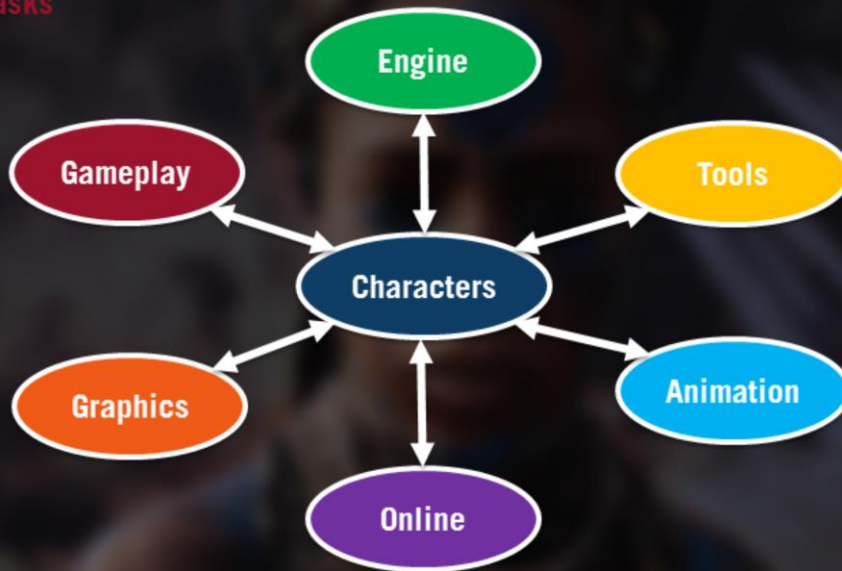


WHAT WENT WRONG

WHAT WENT WRONG

WHAT WENT WRONG

Transversal tasks



FARCRY
PRIMAL

[Julien]

We find that AAA teams are often not equipped to deal with transversal tasks. Changes on the characters may impact all these teams very quickly. If from the conception phase you involve all these parties in the decision making, it becomes really hard to reach an agreement, simply because of the number of people involved.

For Primal, a tight scheduled meant we had to reduce the number of stake holders to 1 lead programmer and clarify the approval process with a short term deadline.

Another thing we did, was to organize development a bit differently than usual. Instead of having engine team, tools team, pipelines etc collaborating, we focused the subject of character tech onto one programmer (me!). That way we avoided cells waiting for each other and conflicting schedules causing bottlenecks. It also allowed for a single point of contact with full knowledge for the users.

WHAT WENT WRONG

Lost knowledge



FARCRY
PRIMAL

[Julien]

Another issue came from legacy.

Dunia is based on the version of CryEngine Ubisoft bought from Far Cry a dozen years ago. Over a decade of developments with a huge number of programmers contributing and leaving the team, we find ourselves in the situation where we have lost knowledge on some of the systems. This caused a difficulty in evaluating tasks relating to anything in the pipeline, which caused fear of change, enabling the pipeline to grossly overstay its welcome.



[julien]

Where might this pipeline will go in the future?

Obviously we're not here to make any announcements but let's think at ways we could push things forward.



[julien]

Wolfskin is data oriented and doesnt know it's building characters, so it could be applied to any entity built with parts. In past games, we have had a focus on animals, vehicles, weapons. These all seem like the type of stuff Wolfskin could handle well.

FUTURE



[julien]

We would also like to extend the pipeline to MotionBuilder. That way, animators would be able to always have the latest version of the characters in MoBu, instead of relying on manually setup fbx scenes like now.



Special thanks to Julie Truong and David Robillard for greenlighting Wolfskin on Primal, Thomas Felix from Passenger and Marc-André Ferland from Far Cry for supporting the pipeline, Arnaud Kotelnikoff our lead character artist, and the Far Cry Primal team.

Julien.Lallevé@ubisoft.com
Character Tech Programmer
@jlallevé

Kieran.OSullivan@ubisoft.com
Character Technical Director
@kizzafreak

QUESTIONS?

