Alembic Workflows

RenderMan for Maya improves Alembic rendering workflows by consolidating existing functionalities.

There are now different ways to render Alembic archives in RFM. We will review them one by one.

This documentation uses an Alembic archive of the **Rolling Teapot** model from **Brice Laville Saint Martin**.

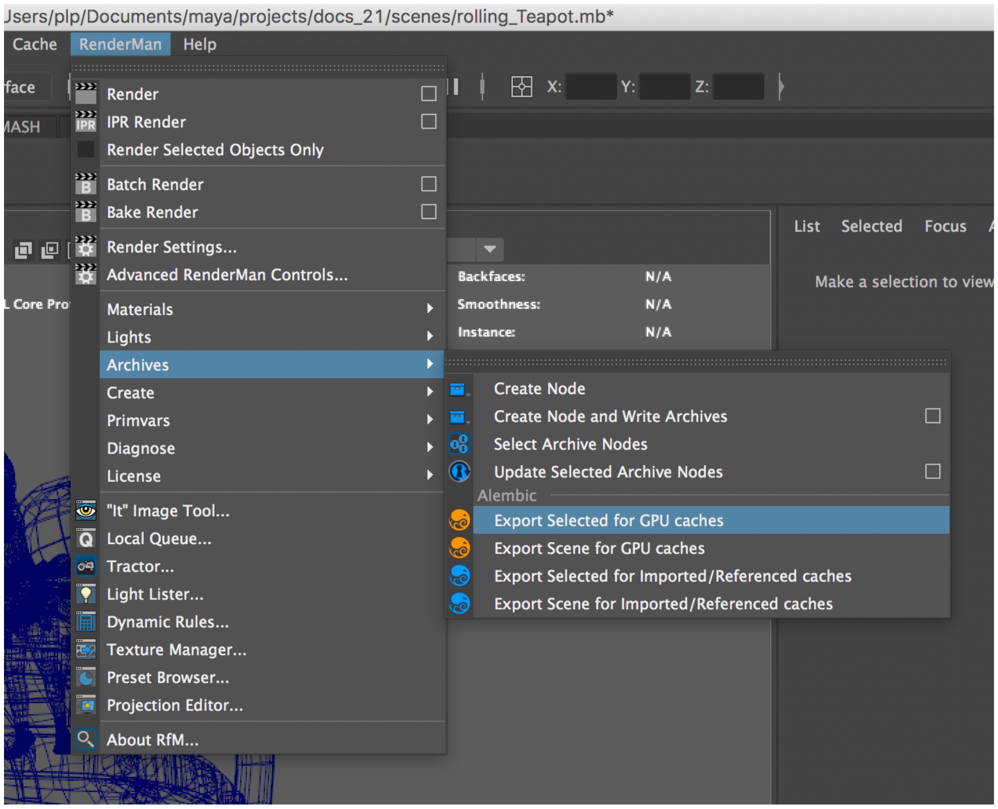
You can download it **here**.

Exporting geometry to alembic from Maya

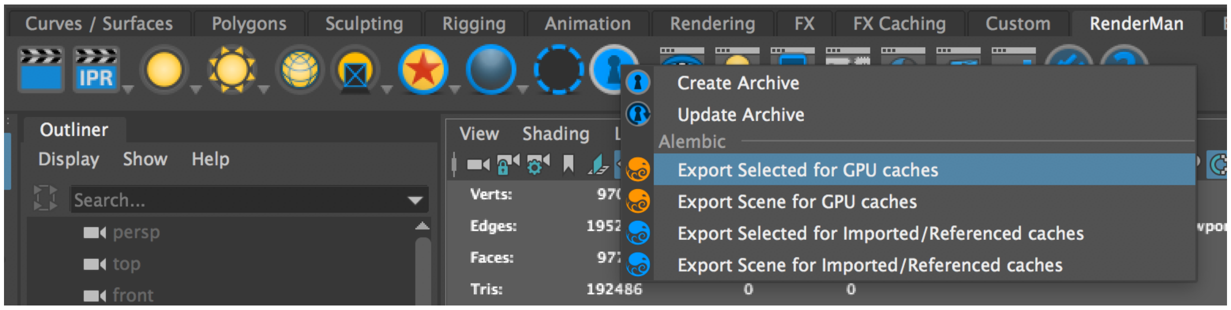
First, you need to export your geometry so that RenderMan may render it correctly.

To make it easier, we have added a few export functions in RFM to guarantee correct results, but you can roll out your own based on our work if you need to customize it.

You can find it in the Archive section of the RenderMan menu...



...or by right-clicking the archive icon in the RenderMan shelf.



As you can see, there are different options depending on how the archive will be rendered:

Export for the gpuCache node

When exporting for the gpuCache node, the alembic archive will be sent directly to the renderer, without being processed by RfM. If your scene contains polygon meshes that are either smoothed (using the Smooth Mesh controls) or carry RenderMan's subdivision surface attributes, we need to make sure they will be exported as subdivision surfaces instead of poly meshes. This menu includes a pre-processing step before calling the usual Alembic export dialog.

> We use the **gpuCache** node to reference, display and render regular, high quality Alembic archives.
>
> It is different from Maya's GPU cache workflow where you save a simplified version of your scene, simply to keep large scenes more interactive.
>
> Always use our Alembic Export menus for best results. If you try to render alembic files saved via the Cache > GPU Cache > Export... menu, we can not guarantee the results.

Export for imported/referenced archives

When an alembic node is imported or referenced in a Maya scene, the archive's nodes will be rebuilt as Maya shapes and must carry their original RenderMan attributes to render as expected. Our export menus include a setup step to make sure all relevant attributes will be saved in the alembic cache, as well as uv sets and subdivision surface creases and corners.

Rendering Imported or Referenced Caches

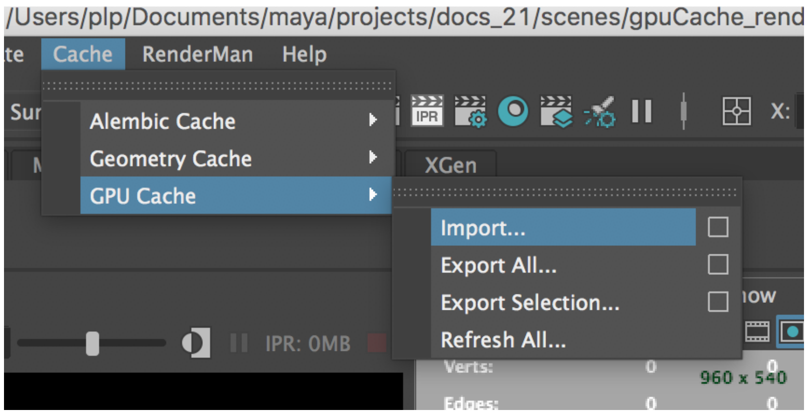
Alembic archives can either be imported or referenced in Maya. You can attach materials to them and they will render exactly like any other piece of Maya geometry.

They are great to light and shade animated characters without the burden of evaluating complex rigs. They are not so great when the archive contains a large number of nodes with heavy geometry, in which case you might consider rendering a GPU cache instead.

Rendering gpuCache nodes

Creating a new gpuCache node

Start by importing your alembic file via the **GPU Cache** menu. If the menu doesn't appear in your Maya, go to the **Plug-in Manager** window to enable it.

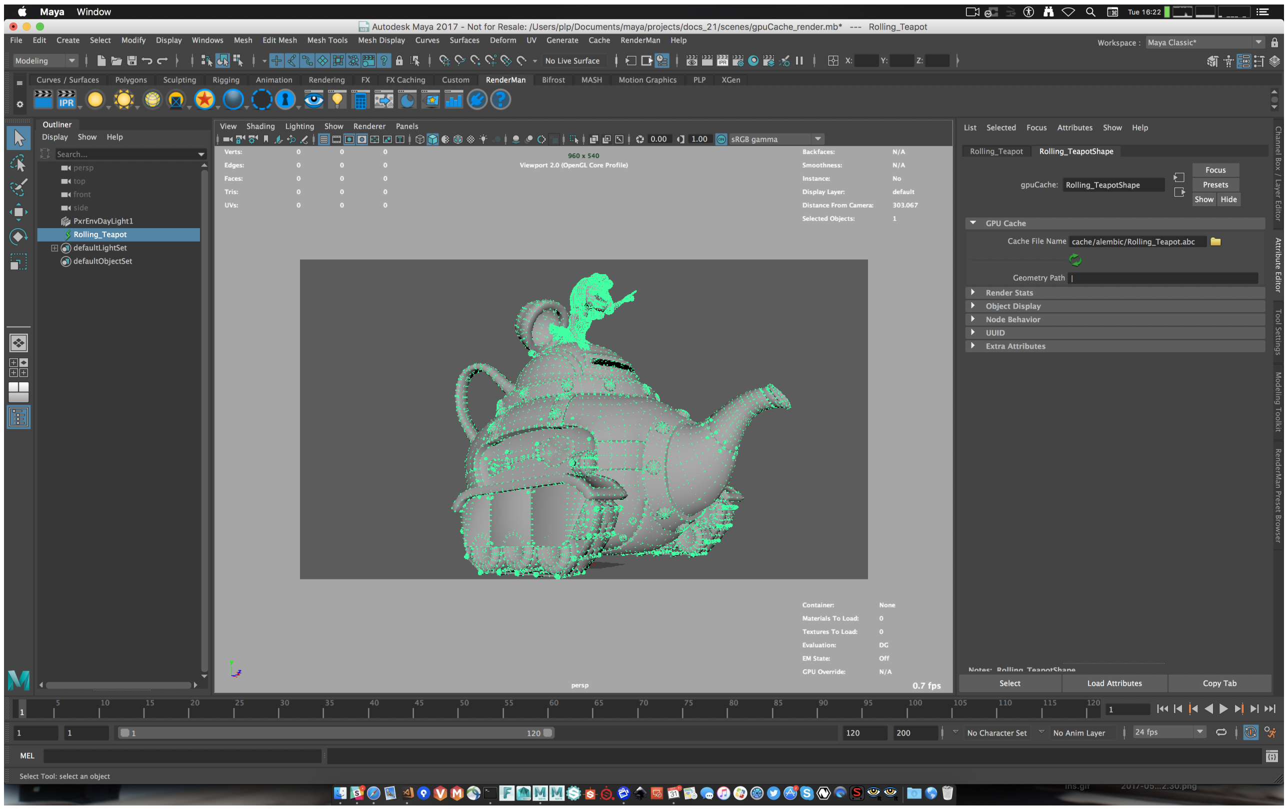


By default, a gpuCache node is invisible to indirect rays !

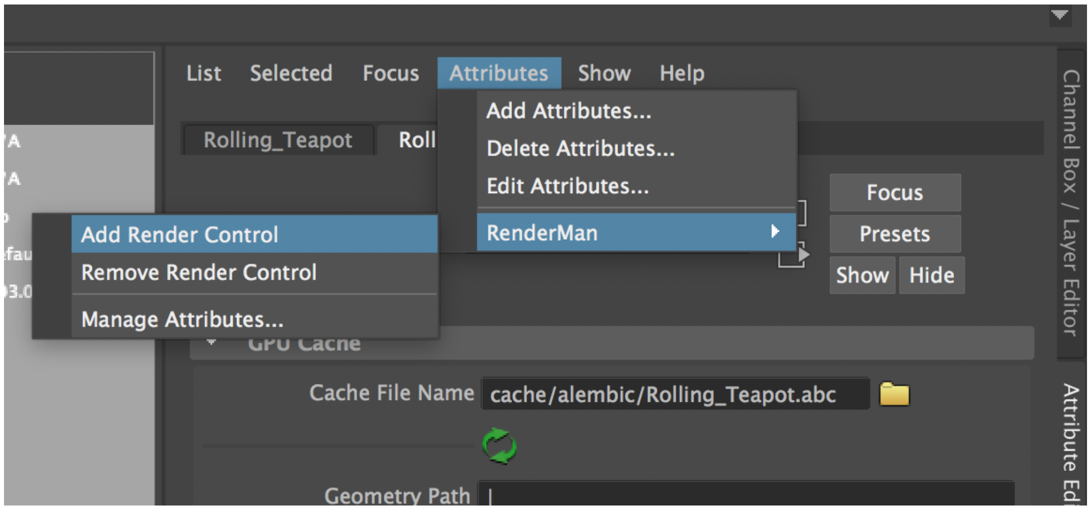
Go to the **Render Stats** section in the Attribute Editor and enable **Visible in Reflections**.

For animated GPU caches please add: Attributes > RenderMan > Manage Attributes > Evaluation Frequency. Set this to Every Frame or your render will fail to animate correctly.

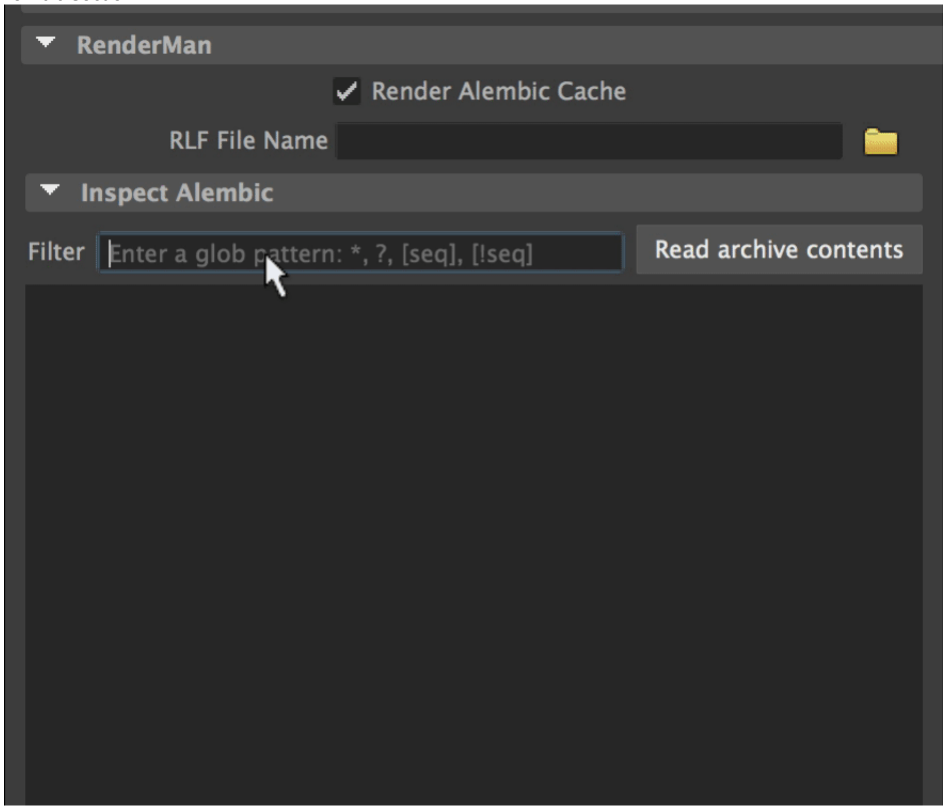
In this example, I imported the Rolling Teapot model. Note that the whole model is now a single node in Maya, which by default will not be rendered by RenderMan.



To make the GPU cache renderable, we need to add special RenderMan attributes. Select the gpuCache node, go in the Attribute Editor's **Attributes** menu and select **Add Render Controls.**



The default attributes will appear in the RenderMan section:

- Render Alembic Cache
  - MUST be enabled to render.
- RLF File Name
  - The full path to a RenderMan Look File that will be applied to the Alembic file.
- Inspect Alembic section
  - 

  - Read archive contents

    This button to parse the contents of the archive and display it the node lists.

    Large alembic archives may take many seconds to parse. Be patient... 🙂

    The node list may incorrectly display large number of deeply nested nodes. To work around this Maya bug, we only expand the first 3 levels by default and suggest using the filter field to explore the archive.

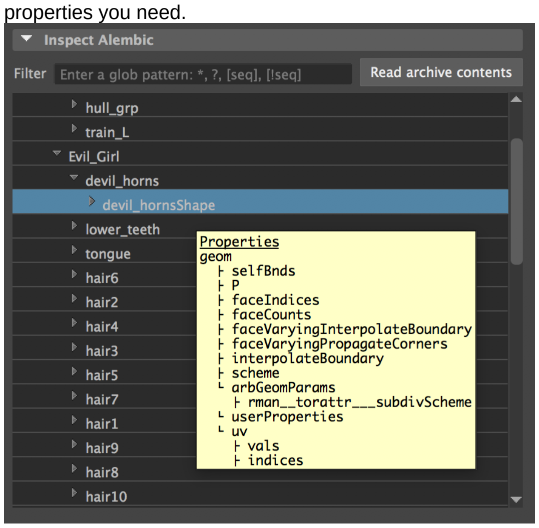
    The parsed data is saved on the node so it may be displayed quickly without having to re-parse the file, but you will have to re-parse the archive if anything has changed in it.

  - Filter
    - The contents of the node list can be filtered using a glob expression.

Pattern	Meaning
*	matches everything
?	matches any single character
`[seq]`	matches any character in *seq*
`[!seq]`	matches any character not in *seq*

  - Node list
    - A hierarchical view of the archive's contents.

- Each node will display its properties as an annotation. This is useful to check if your archive contains the properties you need.


Inspect Alembic

Filter: Enter a glob pattern: *, ?, [seq], [!seq] — Read archive contents

```
    hull_grp
    train_L
  Evil_Girl
    devil_horns
      devil_hornsShape
    lower_teeth
    tongue
    hair6
    hair2
    hair4
    hair3
    hair5
    hair7
    hair1
    hair9
    hair8
    hair10
```

```
Properties
geom
  ├ selfBnds
  ├ P
  ├ faceIndices
  ├ faceCounts
  ├ faceVaryingInterpolateBoundary
  ├ faceVaryingPropagateCorners
  ├ interpolateBoundary
  ├ scheme
  └ arbGeomParams
     ├ rman__torattr___subdivScheme
  └ userProperties
  └ uv
     ├ vals
     ├ indices
```

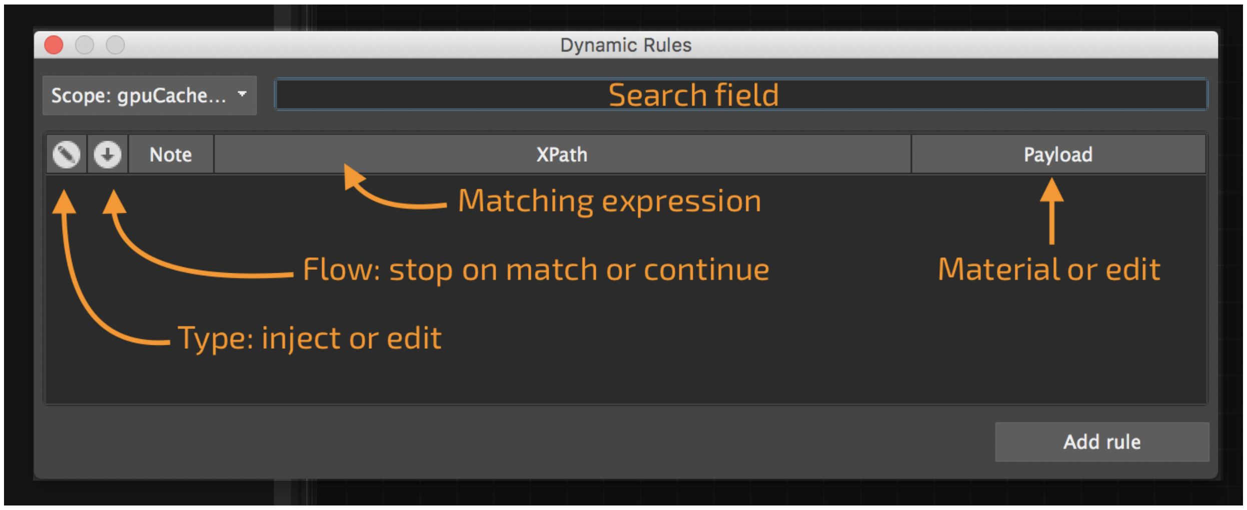
Shader Assignments

A gpuCache is a single shape in Maya, so it is impossible to assign materials as usual. But we can use RenderMan's **Dynamic Rules Editor** to assign materials at render-time, with the following limitations:

- IPR material edits don't work when using Dynamic Rules
- Any rule change will require an IPR restart to take effect.
- These limitations will be lifted in an upcoming release.

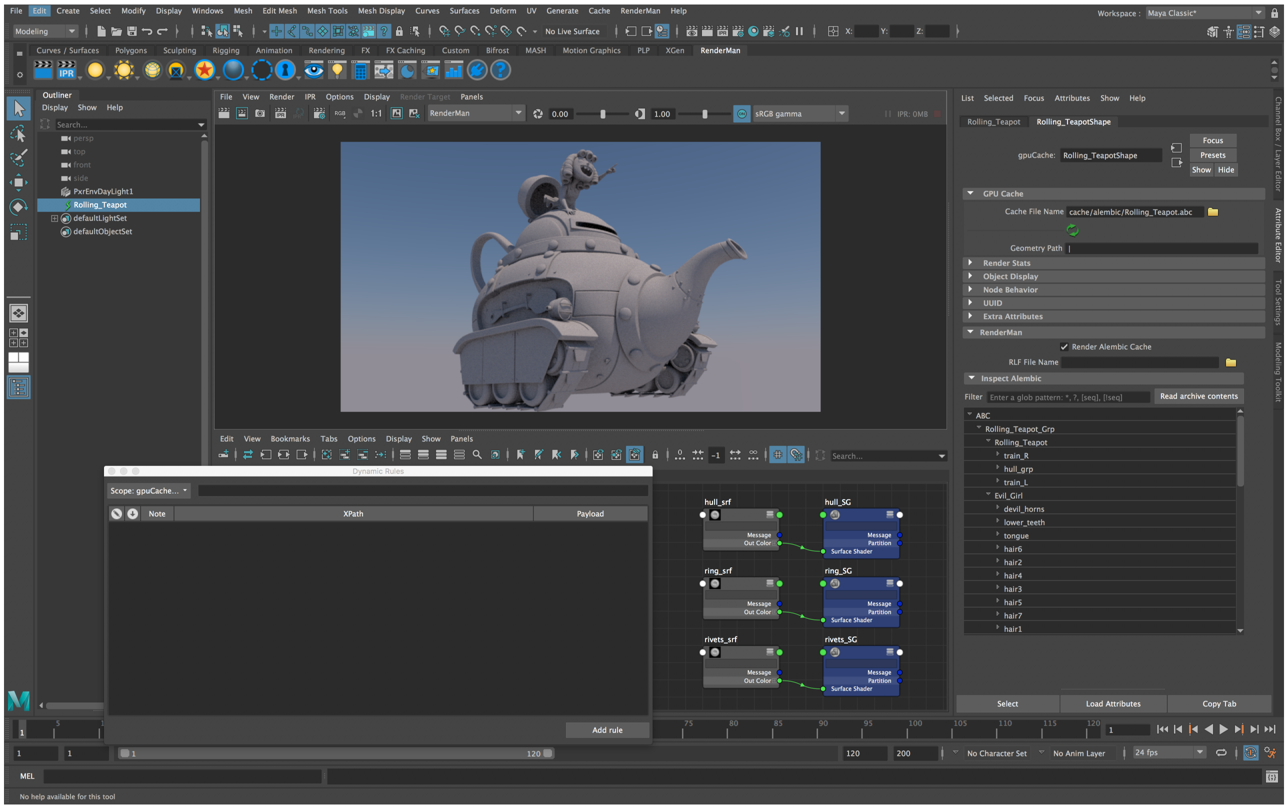
In-scene shading

Open the Dynamic Rules Editor (in the RenderMan menu or the calculator icon in the shelf) and let's see how it works.



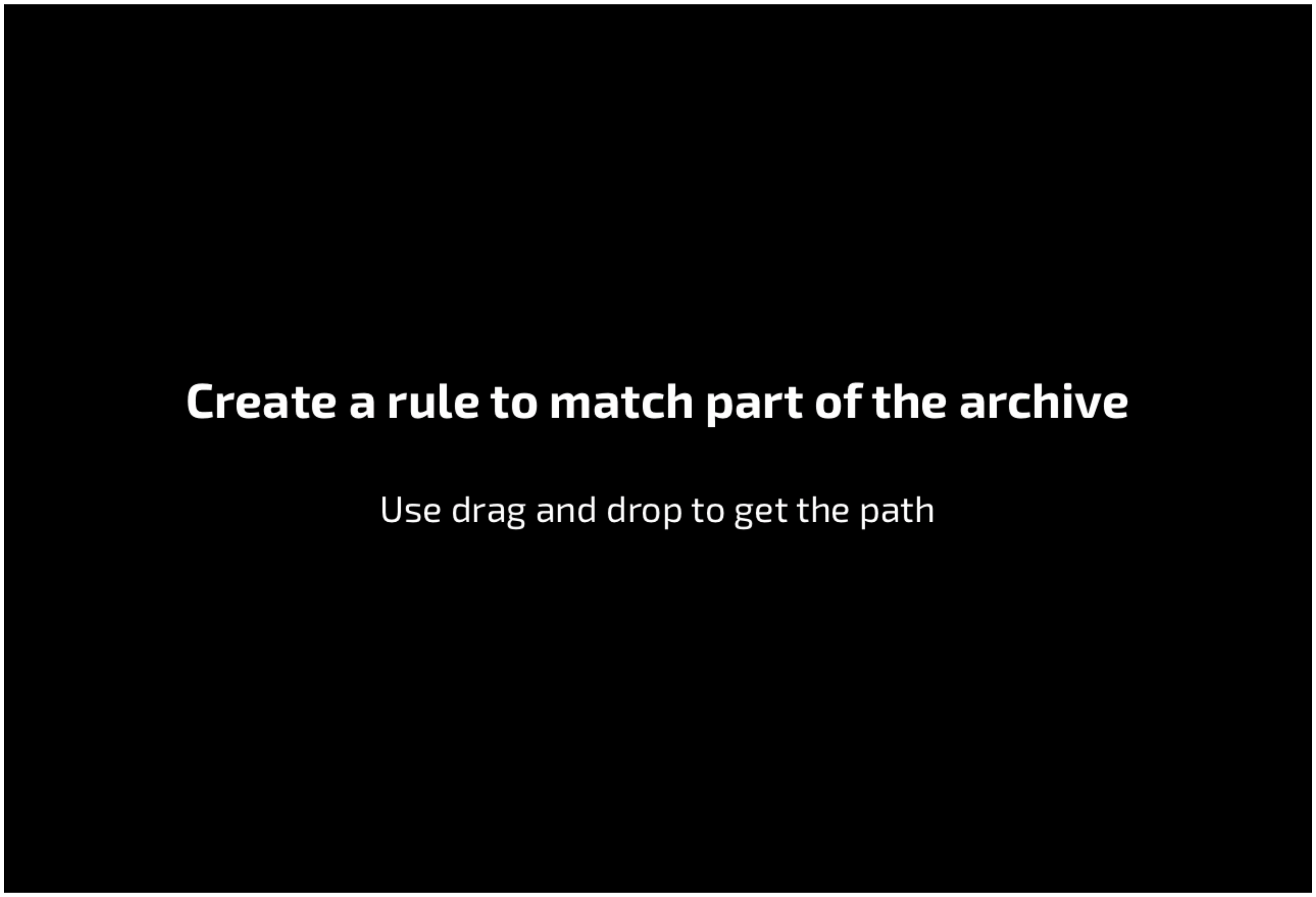
Dynamic rules are written using XPath. See the Dynamic Rules CookBook below for more infos.

This is our starting point: the gpuCache node renders with the default material:



Assign a base material to part of an archive

We have prepared a number of basic materials to assign. Let's start by assigning the `hull_srf` material to the whole teapot and write our first rule:



Create a rule to match part of the archive

Use drag and drop to get the path

- In the node list, select the top node of the hierarchy we want to assign to: Rolling_Teapot.
- Click **Add Rule** to create a new rule
- Double-click the XPath field to put it in edit mode.
- Go to the Alembic node list and middle-mouse-drag the selected node into the expression field.
    - Please note that you must select an item in the list before drag-and-dropping.
- Edit the expression:

```
//Rolling_Teapot_Grp/Rolling_Teapot//*
```

    - All expressions must start with `//` to ignore obsolete RfM render passes.
        - A double slash is generally equivalent to `/*/` in a shell.
    - Add `//*` at the end to match all shapes at the end of matching paths.
    - Press enter to validate.
- Double-click the **Payload** field and select `hull_srf`.
- Launch the IPR to test.

Match sub-strings in a hierarchy

Now we will shade the rivets on the teapot. We will need a more complicated expression.

Create a rule for rivets

Duplicate and edit the first rule

- Duplicate the first rule
    - Right-click on the rule number (0 in our case) and select **Duplicate Rules**.
- Extend the previous expression to match any object containing the word "rivet".

```
//Rolling_Teapot_Grp/Rolling_Teapot//*[contains(name(), 'rivet')]//*
```

    - `[...]` means we will define a sub-expression
    - `contains(path, substring)` matches if substring is in path.
    - `name()` returns the current node's name.
    - Finally, we extend the expression with `//*` to match shape nodes, as in the previous expression.
- On that model, there are also screws that could use the same material. The expression can be extended to match them too:

```
//Rolling_Teapot_Grp/Rolling_Teapot//*[contains(name(), 'rivet') or contains(name(), 'screw')]//*
```

Rule execution order

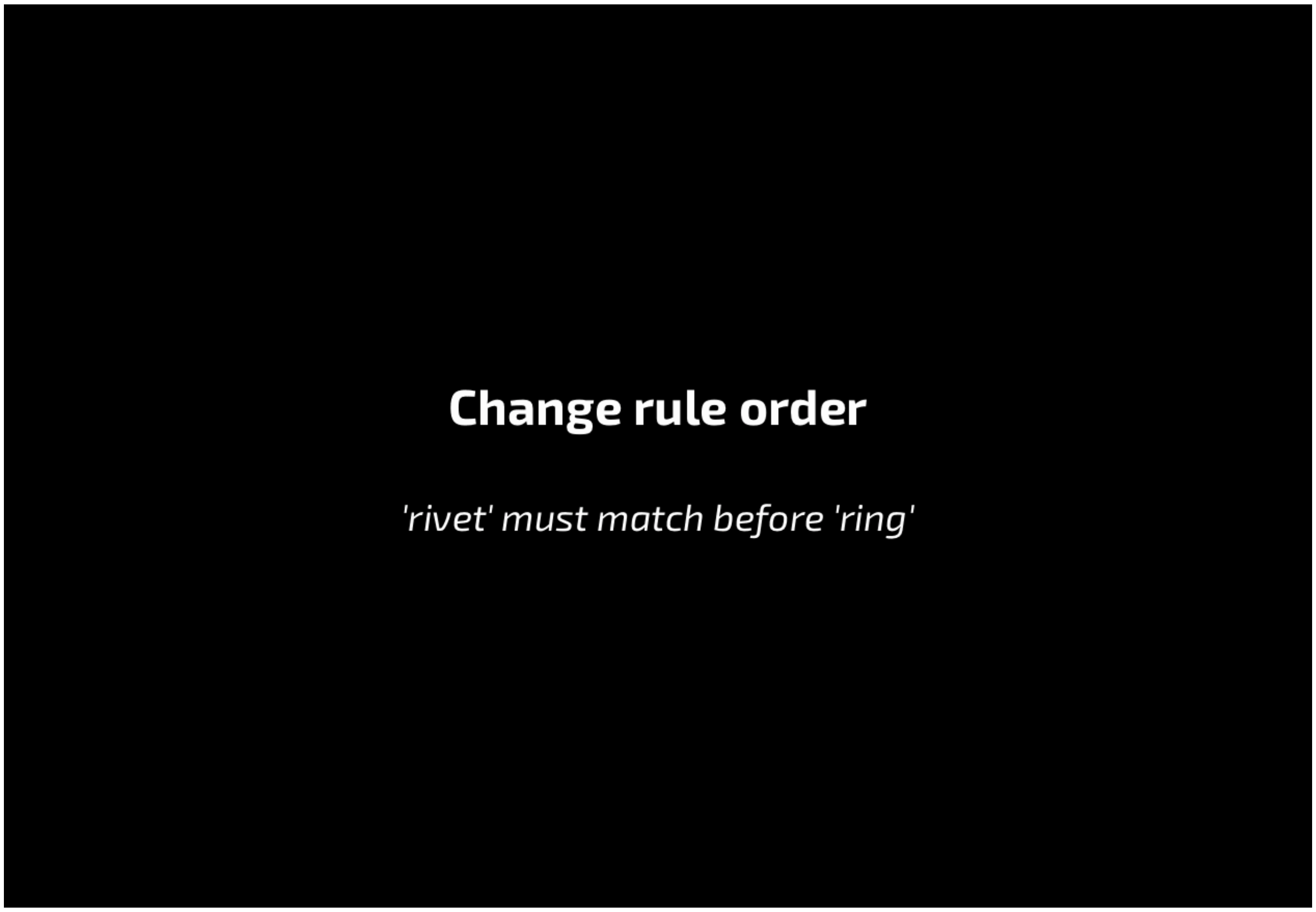
Let's assign a material to the teapot's rings.

We use the same workflow: duplicate the rule, replace `"rivet"` with `"ring"` and set the Payload to `ring_srf`. But it didn't quite work because, as you can see in the node list, some shapes have both `"rivet"` and `"ring"` in their name and rivets on the rings are now using the wrong material.

Here is what happens:

- Rule 0 matches shapes with `"ring"` and then stops, because the **Flow** mode defaults to **Stop on Match**.
  - If a shape is named `"turret_ring_rivet2"`, it will get the ring material, although it is a rivet.
- Rule 1 matches shapes with `"rivet"` that were not matched by rule 0.
  - `"turret_ring_rivet2"` will not be matched because the rule evaluation stopped with rule 0.

We can fix this by changing the **rule execution order**: if we match rivet first, the matching engine will stop when it finds a match and then only shapes only containing "ring" will match the second rule.



- Position the cursor over the rule's number and left-button-drag rule 1 (rivets) above rule 0 (rings).
- Render to check.

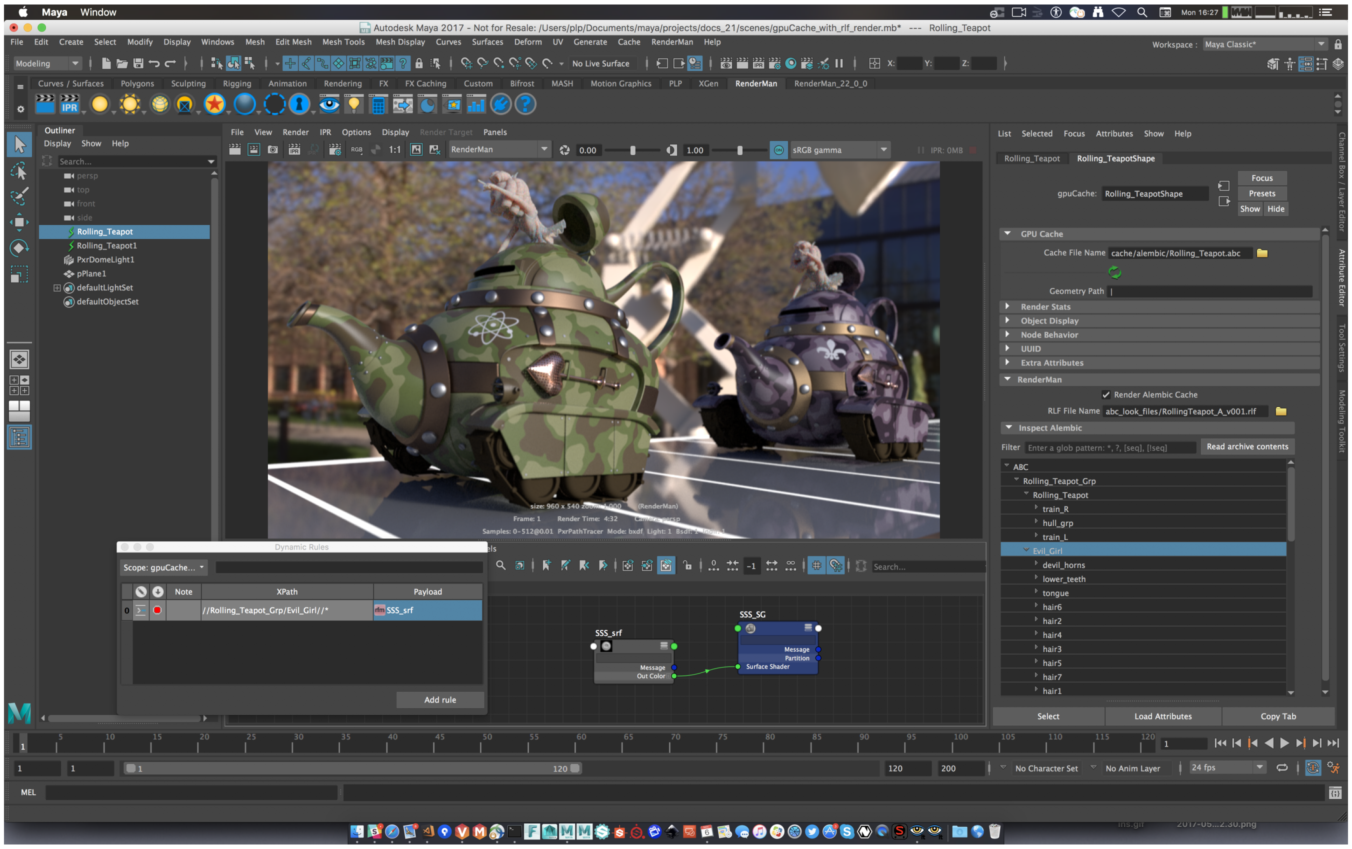
Of course, we have only scratched the surface and we recommend that you read up on XPath for more sophisticated expressions.

Self-contained Asset Shading

If you want to import fully shaded assets, there are 2 options:

- When RfM finds a rlf file in the same location and with the same name as the archive, it will automatically apply it.
  - This is not always very flexible in a pipeline, but may be useful for archiving assets.
- If the **RLF File Name** field of a gpuCache node contains a path to a rlf file, RfM will apply it to that gpuCache node.
  - It becomes easy to script look assignment and updating.
  - We provide a sample RLF export script that can be downloaded here.

Note that you can still override automatic assignments in the scene with the Dynamic Rule Editor !



gpuCache: Rolling_TeapotShape

Cache File Name: cache/alembic/Rolling_Teapot.abc

Render Alembic Cache ✓

RLF File Name: abc_look_files/RollingTeapot_A_v001.rlf

Scope: gpuCache...

Note	XPath	Payload
	//Rolling_Teapot_Grp/Evil_Girl//*	SSS_srf

ABC
- Rolling_Teapot_Grp
  - Rolling_Teapot
    - train_R
    - hull_grp
    - train_L
  - Evil_Girl
    - devil_horns
    - lower_teeth
    - tongue
    - hair6
    - hair2
    - hair4
    - hair3
    - hair5
    - hair7
    - hair1

Notes about RLF export

When you reference multiple RFL files in a scene, you must make sure that all shading nodes have unique names. If you define the same name multiple times, the renderer will use the first in line and you will get unexpected results.

In our sample export script, we ask the user to provide a Unique Identifier string that will prefix all node names in the RLF file.

Dynamic Rules CookBook

Imagine we have a gpuCache node named Rolling_Teapot...

Everything in the scope, i.e. the Maya scene:

```
//*
```

All nodes in the gpuCache node:

```
//Rolling_Teapot//*
```

All nodes of the gpuCache with "rivet" in their name:

```
//Rolling_Teapot//*[contains(name(),'rivet')]//*
```

All nodes of the gpuCache with "rivet" or "screw" in their name:

```
//Rolling_Teapot//*[contains(name(),'rivet') or contains(name(), 'screw')]//*
```

All nodes of the gpuCache with "rivet" but not "screw" in their name:

```
//Rolling_Teapot//*[contains(name(),'rivet') and not(contains(name(), 'screw'))]//*
```

All shapes of the gpuCache under a transform starting with "ring_":

```
//Rolling_Teapot//*[starts-with(name(),'ring_')]//*
```

Match 2 sub-groups in the cache's hierarchy:

```
//Rolling_Teapot//*//train_L//* | //Rolling_Teapot//*//train_R//*
```