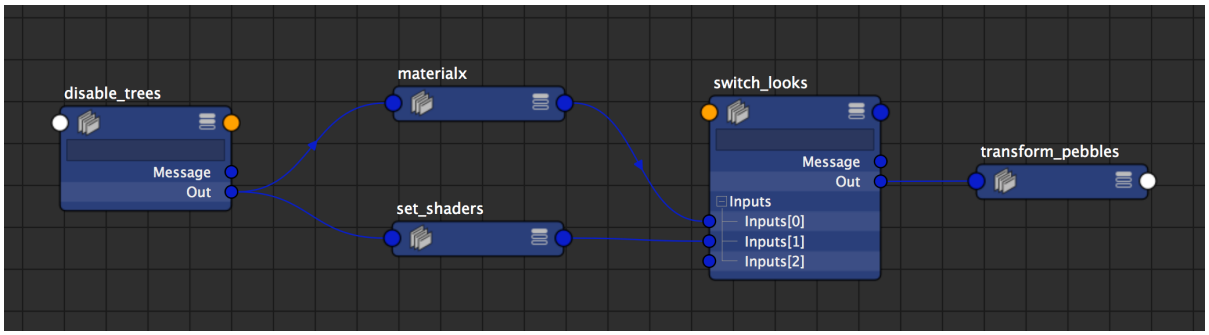
Operators



Operator graph in the Node Editor

> A video that shows rendering workflows with Arnold *Operators* can be found here.

Operators allow advanced users to override any part of an Arnold scene and modify the Arnold universe at render time. Probably one of the most common use cases is to override parameters (e.g. shaders) inside a procedural (e.g. ASS or Alembic). To achieve this you must know the Arnold node and parameter names defined inside the procedural.

Operator nodes perform per-object (node) parameter assignments and overrides, including late-bindings and deferred overrides on procedurally generated nodes. *Operators* can also create nodes (MaterialX) and carry out general scene inspection and modifications.

Some *Operators* provide a selection parameter that determines, using a wildcard expression, what nodes are processed by the operator. This is discussed in more detail in the section on selection expressions below. If an operator is being evaluated with regards to a procedural it's connected to the selection expression is assumed to be relative to the procedural's namespace (see operator graphs below).

Operators can be chained together in an operator graph which is evaluated from a given target operator. Multiple disconnected operator graphs can exist in the scene, where only the graph connected to the target operator and operator graphs connected to procedural nodes will be evaluated for rendering.

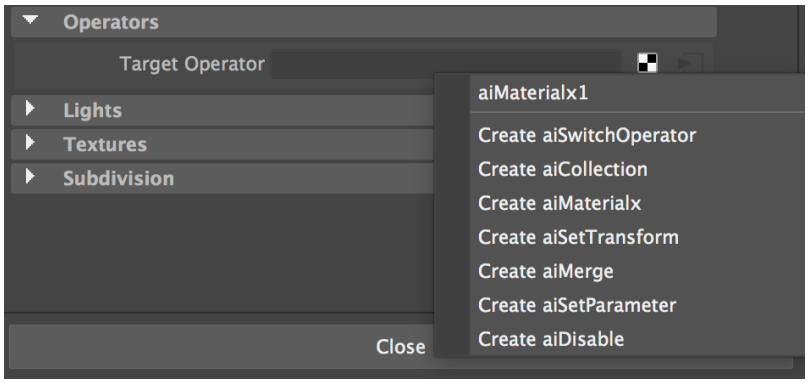
> Kick can be used to query possible target node parameter names of a set_parameter node. For example:

```
Kick -info <node type>

Kick -info polymesh
```

Operators can be ignored in the Arnold render settings by ticking *Ignore Operators* in the *Diagnostics* tab.

Tutorials about *operators* can be found here.



Setting the target operator in the Render Settings

Selection Expressions

An operator can use a selection to determine what nodes will be selected to be processed by the operator at render time. The selection is an expression that can consist of multiple selections concatenated by:

- or (union)
- and (intersection)
- not (negation)
- and not (exclusion)
- () for nested scoping

Each selection uses glob and regex expression to determine if an operator processes a given node, where a node will be processed by the operator if the expression matches the node name. By default glob matching is used unless the selection is in a regex quote, i.e. r'[regex]'.

- `(/group0/sphere* and not (/group0/sphere1 or /group0/sphere0)) or /group1/sphere3`
- `not r'p(ickle|ringle)[0-9]+'`
- `r'c(ar1|ar2)' or r'car[34]'`

Parameter Matching

Selections can also be used to match parameter names and values on the selected nodes, including the node entry name, type, and derived type. This is done using a parameter dot-delimiter `.()` on each node selection string. The following example selection matches all nodes named 'sphere' which have a 'radius' parameter.

- `sphere.(radius)`

Comparators can be used to match certain parameter values. The following selection matches all nodes whose name starts with 'sphere' and has a 'radius' larger than 0.5.

- `sphere*.(radius > 0.5)`

The parameter matching also supports concatenation and glob/regex, e.g.

- `car*.((make == 'fiat' and year > 2010) or tinted_glass == True) or drone*.(battery_level >= 20)`
- `plane*.(model == 'A3*' and captain == r'B(ob|ryan)')`

Arrays and Multi-Value Parameters

Arrays and multi-value parameters such as vectors, RGB, etc. are matched using square brackets. If an array has a single value or if the array consists of single numbers then only one set of square brackets is necessary.

- `*.(rgb_array == [[1 2 3][4 5 6]])`
- `*.(float_array == [10.0 20.0 30.0])`
- `*.(velocity >= [1.0 1.0 1.0])`
- `*.(my_matrix == [1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6])`

String Literals

Matching parameters of type string, enum and node requires string literals, e.g.

- `*.(some_string == '*value*')`
- `*.(transform_type == 'rotate_about_center')`
- `*.(shader == 'purple_shader')`
- `*.(shader == 'yellow_*')`

Note: The value will be treated as a parameter reference if the string quotes are omitted (see below).

Parameter Array Indices

It is possible to match specific array indices in array and multi-value parameters. Square brackets are not necessary when matching a single value.

- `*.(accessories[3] == 'monkey fists')`
- `*.(my_rgb[1] == 1.0)`
- `*.(rgb_array[1] >= [0 0 0])`
- `*.(float_array[1] == 20.0)`
- `*.(rgb_array[1]) # checks if the entry exists`

Matching Parameter References

Commonly, the node already has some arbitrary user parameters coming from e.g. simulation or even another operator.

It is possible to match other node parameters of the same type, e.g.

- `*.(radius <= some_float_param)`
- `*.(my_rgb == some_rgb_param)`

Node parameters can match single string values, e.g. to match a shader based on a user string parameter

- `*.(shader == some_string)`

We can also match parameters on other nodes, where the syntax is #node_name.param_name1[.param_nameN]

- `*.(model == #some_node.model)`

- `*.(year == #some_node.some_int)`
- `*.(year == #some_node.node_array[0].year)`
- `*.(my_rgb == #some_node.some_rgb)`

It can be useful to match values on parameters that are linked to the node. Shaders are a common example, where the value can also refer to other parameters

- `*.(shader.base == 0.8)`
- `*.(shader.base_color == [1 1 0])`
- `*.(shader[0].base_color.filename == '*plate*')`
- `*.(shader[0].base_color.filename == #some_node.tex_name)`

Matching Multiple Parameter Names

We can also match more than one parameter by using a glob or regex expression in the parameter name.

A simple example is matching an RGB parameter regardless of if it's called *color* or *colour.*

- `*.(colo*r == [1 0.4 0.2])`
- `*.(mod* == r'(X|M)[0-9]')`
- `*.(r'receive_sha.*' == True)`

The operator is given all the parameters that matched where it can either use all of them or decide what to do with each one.

Matching Node Entries

The selection can filter based on node entry information such as node entry name (@node), type (@type), and derived type (@derived)

- `*.(@node == 'polymesh')`
- `*.(@type == 'shape')`
- `*.(@derived == 'procedural')`

This can be concatenated in the usual way with other parameter selections.

- `*.(@node == 'sphere' and radius > 0.4)`